Engineering Sketch Pad (ESP)

User Training Session 5
UDPs, UDFs, and UDCs

John F. Dannenhoffer, III
john@geocentrictech.com
Geocentric Technologies LLC

updated for v1.28

Overview

- User-defined Primitives (UDPs) and Functions (UDFs)
  - Difference Between UDPs and UDFs
  - Using UDPARG and UDPRIM Statements
- Creating Simple Cross-sections
- Creating a simple NodeBody, WireBody, SheetBody, and SolidBody
- User-defined Components (UDCs)
  - Include-style
  - Function-style
- Homework Exercise

Differences Between UDPs and UDFs

- Users can add their own user-defined primitives (UDPs)
  - creates a single* Body
  - do not consume any Bodys from the Stack
  - are written in C, C++, or FORTRAN and are compiled
  - can be written either top-down or bottom-up or both
  - have access to the entire suite of methods provided by `EGADS`
  - are coupled into `ESP` dynamically at run time
- Users can add their own user-defined functions (UDFs)
  - are the same as UDPs, except they consume one or more Bodys from the Stack

Calling a UDP (1)

- UDPs are called with a `UDPRIM` statement

```
UDPRIM    $primtype $argName1 argValue1 \
                    $argName2 argValue2 \
                    $argName3 argValue3 \
                    $argName4 argValue4
```

- `$primtype` must start with a letter
- At most 4 name-value pairs can be specified on the `UDPRIM` statement
- More name-value pairs can be specified in any number of `UDPARG` statements that precede the `UDPRIM` statement

```
UDPARG    $primtype $argName1 argValue1 \
                    $argName2 argValue2 \
                    $argName3 argValue3 \
                    $argName4 argValue4
```

- name-value pairs are processed in order (with possible over-writing)

Calling a UDP (2)

- For UDPs that read an external file, one can use << to tell ESP to create a file from the following lines, up to a line that starts with >>
- For example:

```
UDPRIM    editAttr  filename <<  verbose 1
        NODE ADJ2FACE tagType=spar tagIndex=1
        AND  ADJ2FACE tagType=lower
        AND  ADJ2EDGE tagType=root
        SET           capsConstraint=pointConstraint1
>>
SET       A  10
```

has two Branches (UDPRIM and SET)

UDPARG and UDPRIM Examples

- The following generate identical Boxes

```
UDPRIM box dx 1 dy 2 dz 3
```

- and

```
UDPARG box dx 1
UDPRIM box dy 2 dz 3
```

- and

```
UDPARG box dx 11 dy 22 dz 33
UDPRIM box dx 1  dy 2  dz 3
```

- and

```
UDPARG box dx 1
UDPARG box dy 2
UDPARG box dz 3
UDPRIM box
```

Return Values from UDPs

- Some UDPs return values to the calling script
- The returned values have names that are prepended by two at-signs (for example: `volume` in the UDP is available as `@@volume` after the `UDPRIM` executes)
- These values stay in effect until overwritten by another UDP (or a UDF or a UDC)

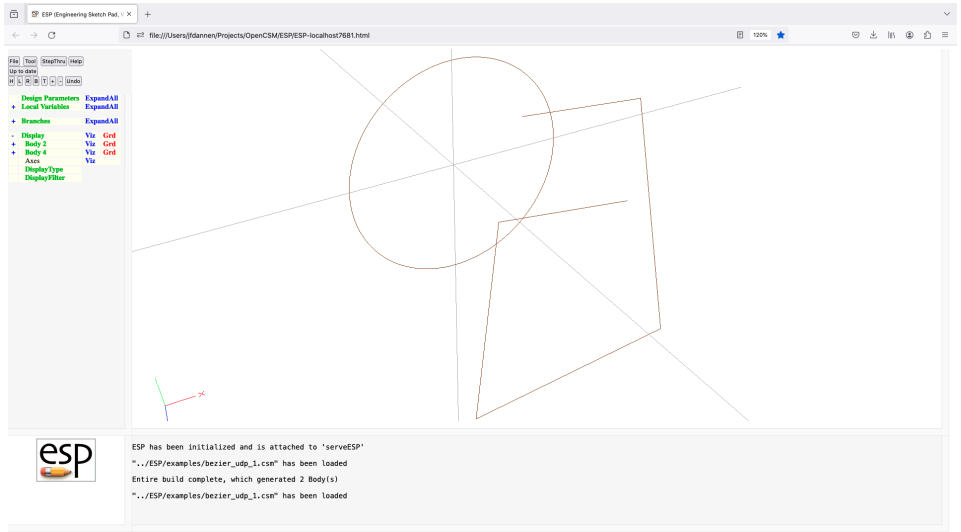
Shipped User-defined Primitives

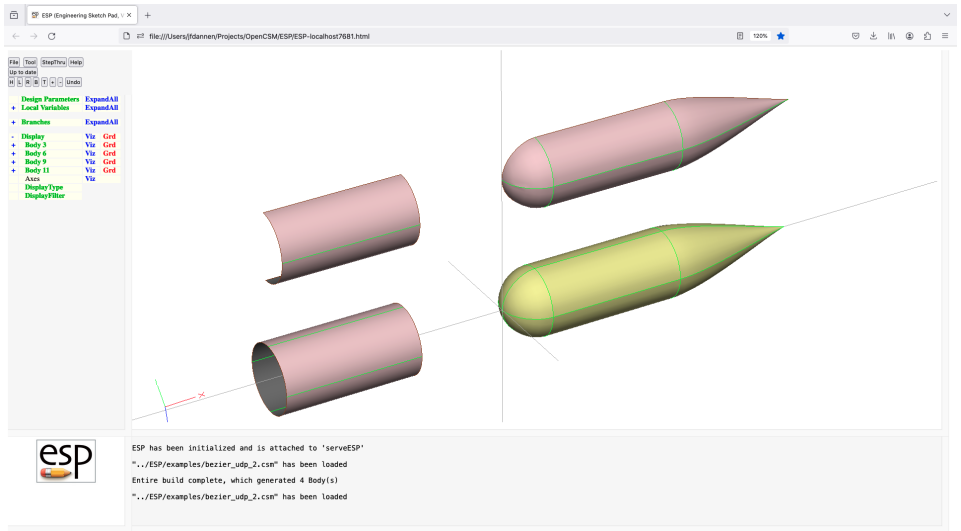
- bezier
- biconvex
- box
- bspline
- csm
- ellipse
- eqn2body
- fitcurve
- freeform
- hex
- import
- kulfan
- naca456
- naca
- nurbbody
- parabaloid
- parsec
- pod
- poly
- prop
- radwaf
- sample
- sew
- stag
- supell
- vsp3
- waffle
- warp

UDP `bezier`

`$filename debug=0 @@imax @@jmax cp[]`

Create a Bezier WireBody, SheetBody, or SolidBody

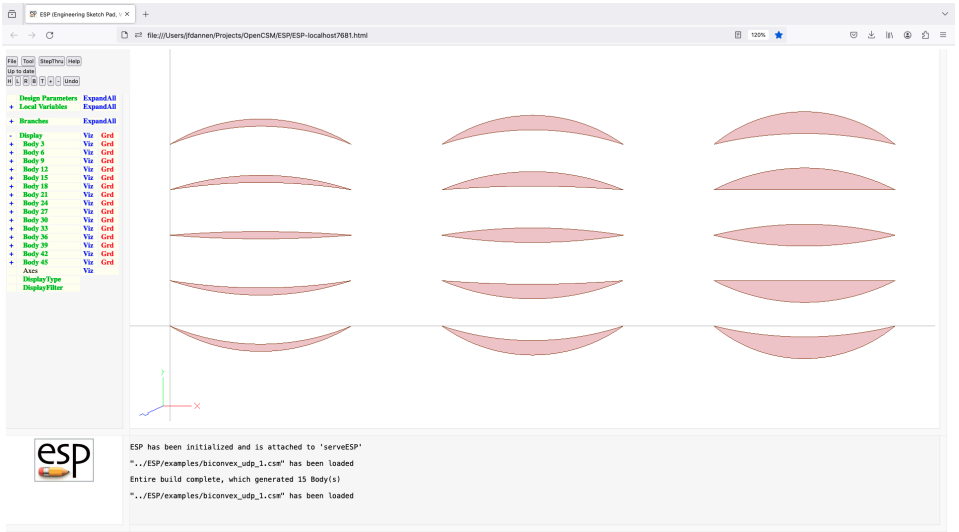




UDP `biconvex`

`thick=0 camber=0`

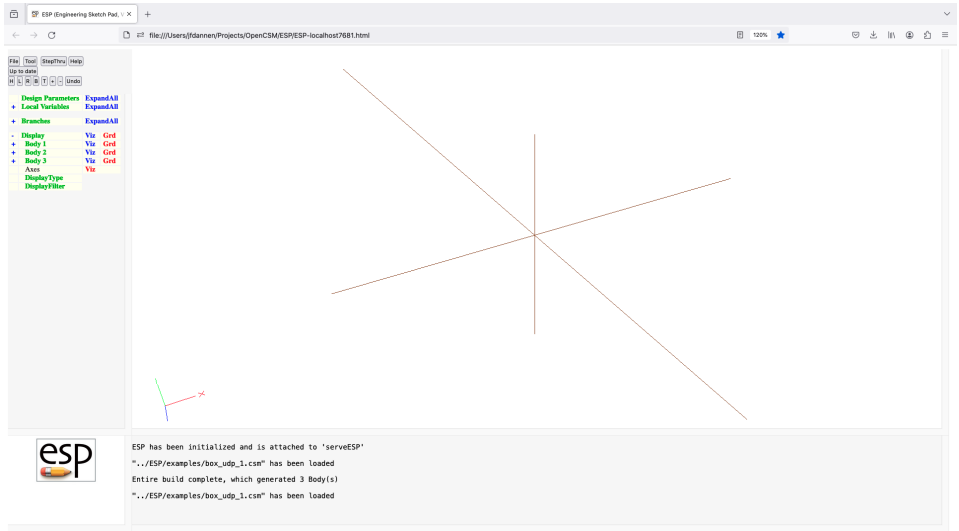
Create a biconvex airfoil SheetBody

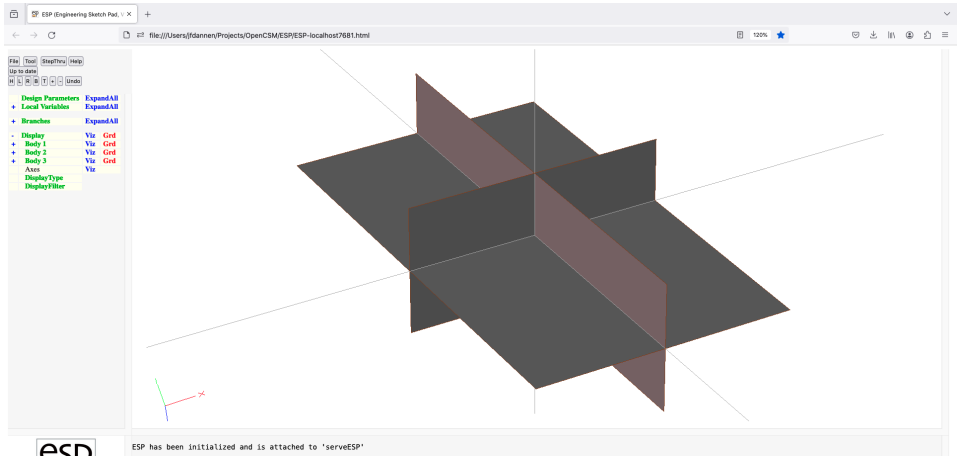


UDP box (1)

dx=0 dy=0 dz=0 rad=0 @@area @@volume

Create WireBody, SheetBody, or SolidBody centered at origin (with possibly-rounded corners)

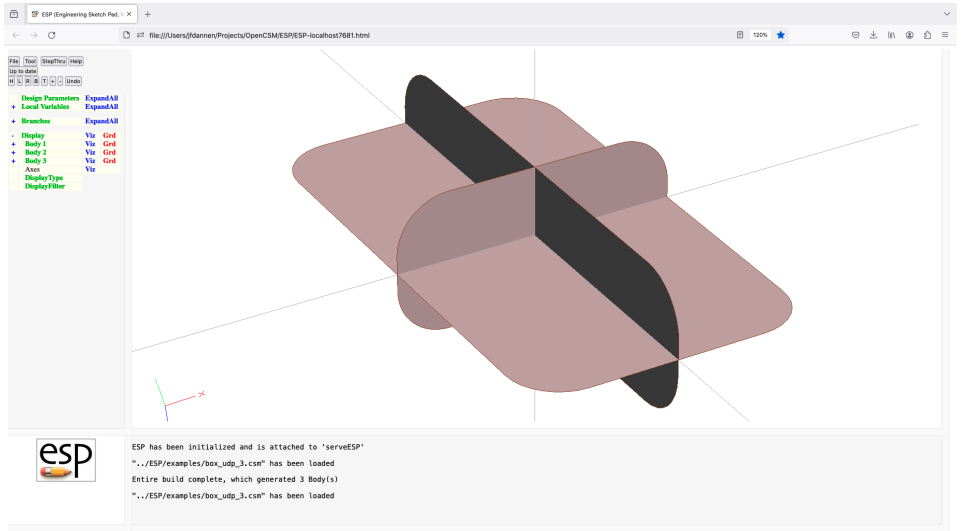


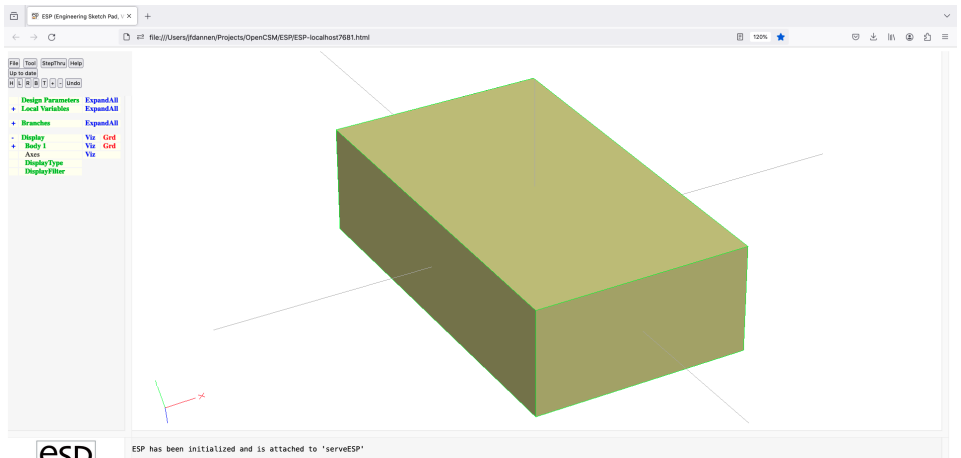


UDP `box` (2)

`dx=0 dy=0 dz=0 rad=0 @@area @@volume`

Create WireBody, SheetBody, or SolidBody centered at origin (with possibly-rounded corners)

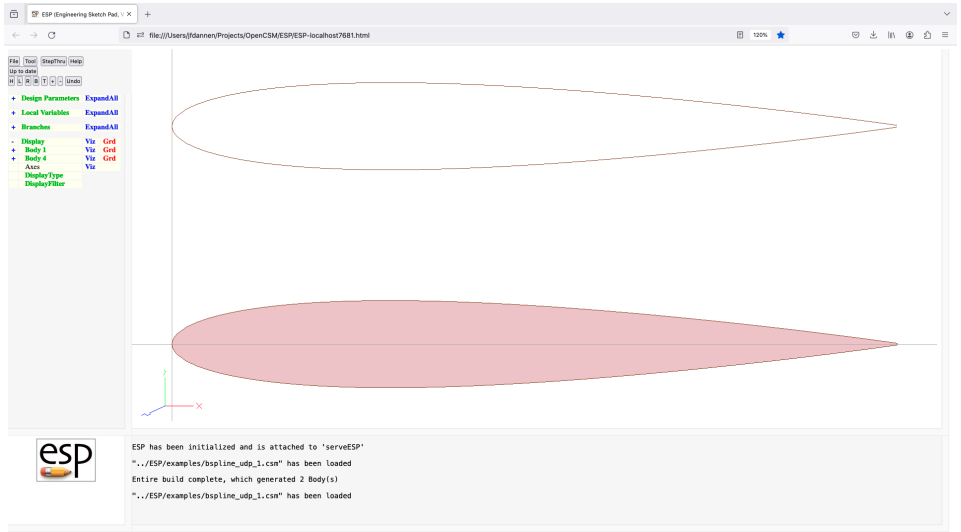


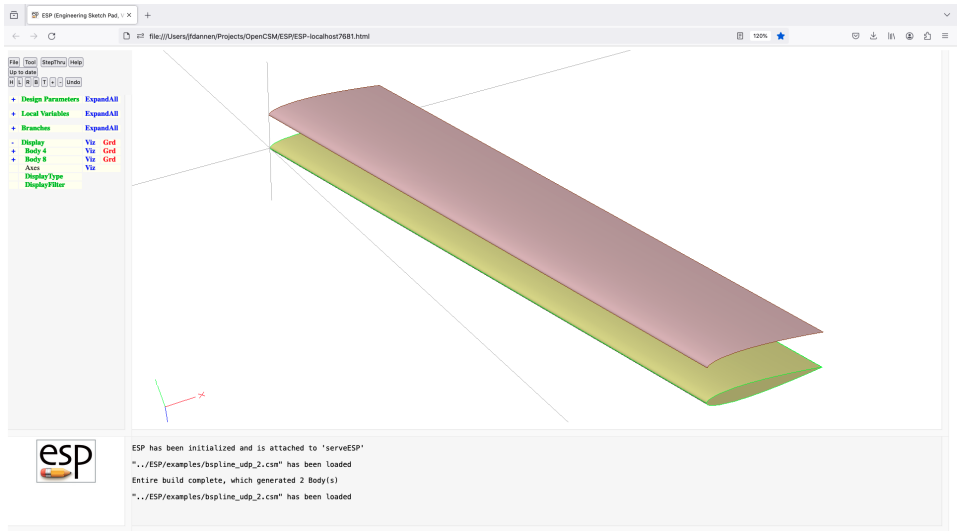


UDP bspline

bitflag=0 uknots[] vknots[] cps[] weights[] udegree=3 vdegree=3

Create a Bspline WireBody or SheetBody

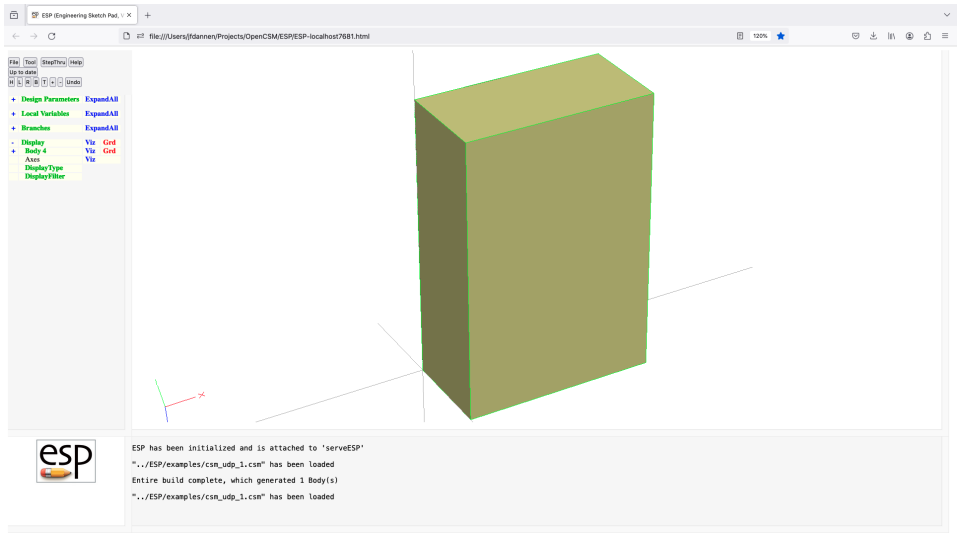




UDP `csm`

$filename $pmtrname pmtrvalue=0 @@volume

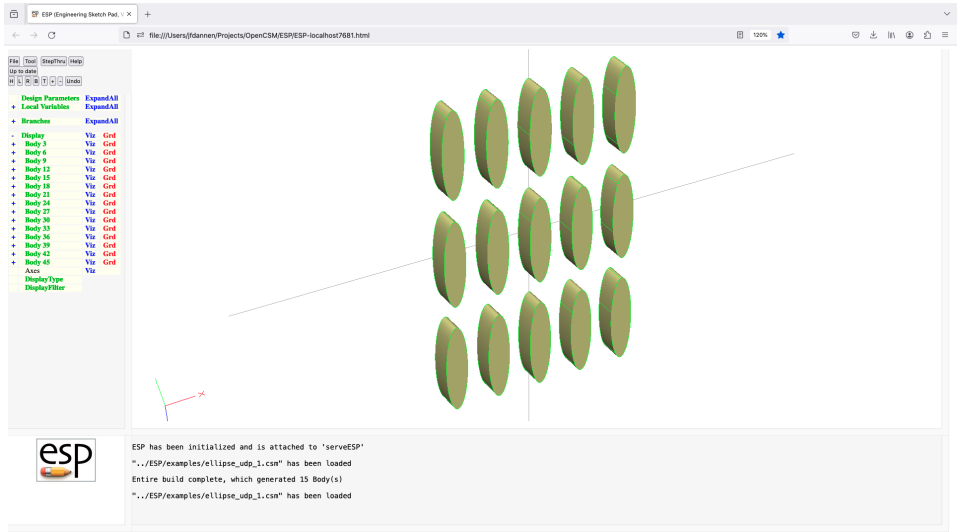
Call `OpenCSM` recursively

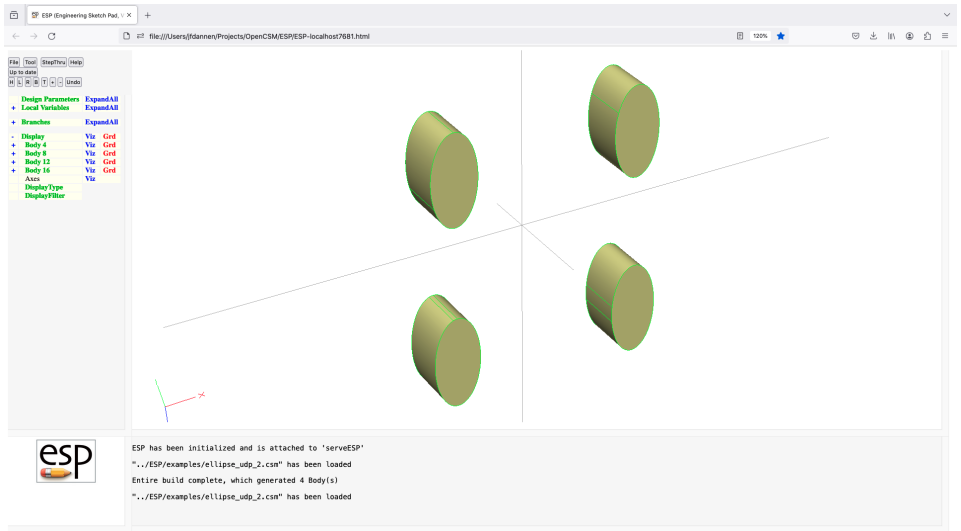


UDP `ellipse` (1)

`rx=0 ry=0 rz=0 nedge=2 thbeg=0`

Create an ellipse SheetBody centered at origin

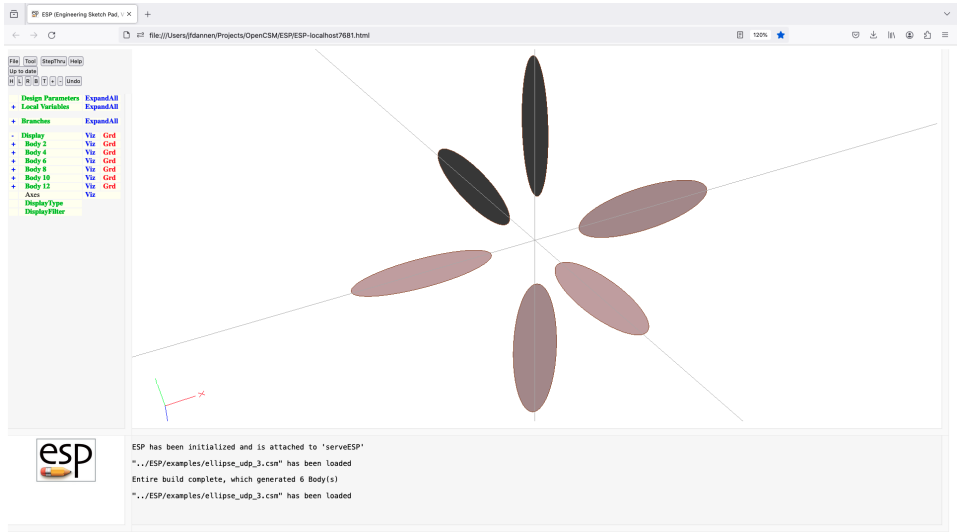




UDP `ellipse` (2)

`rx=0 ry=0 rz=0 nedge=2 thbeg=0`

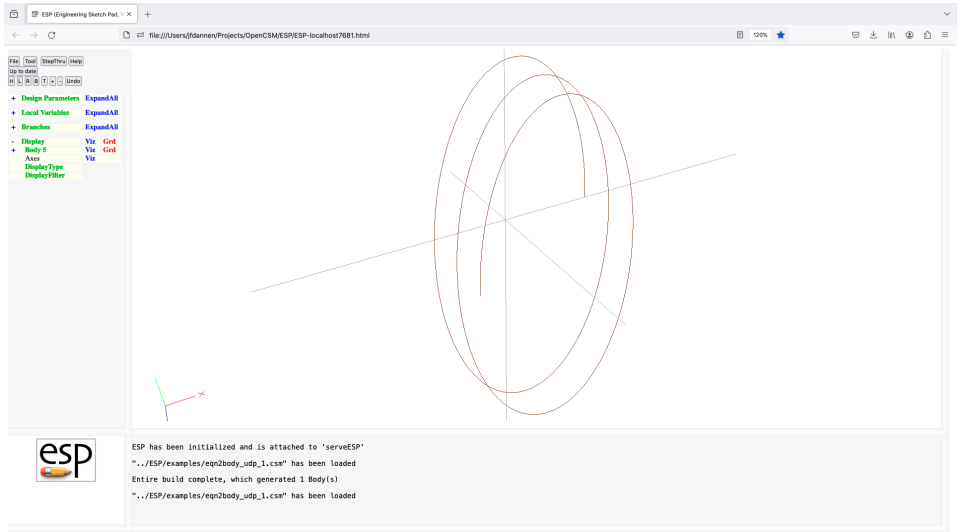
Create an ellipse SheetBody centered at origin

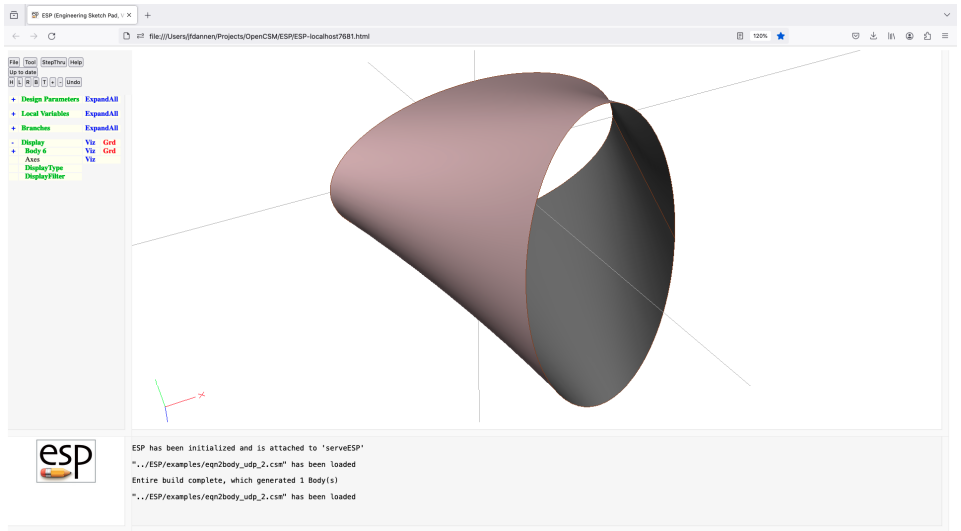


UDP eqn2body

$xeqn $yeqn $zeqn urange[] vrange[] toler=1e-5 npnt=101

Create a WireBody or SheetBody from given equations





UDP `fitcurve`

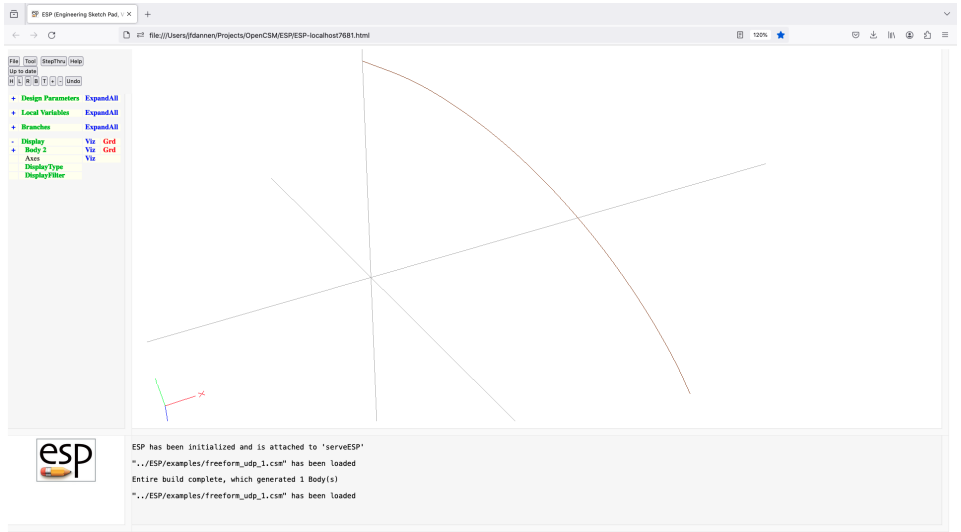
`$filename ncp ordered periodic xform[] @@npnt @@rms`

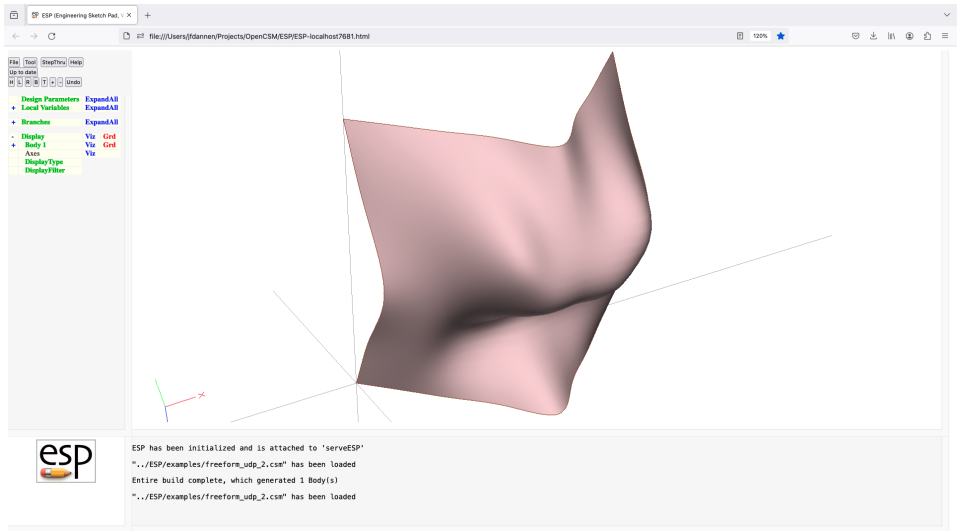
Fit a Bspline WireBody to a set of points (or file)

UDP `freeform` (1)

`$filename imax=1 jmax=1 kmax=1 xyz[]`

Create a freeform WireBody, SheetBody, or SolidBody from a file

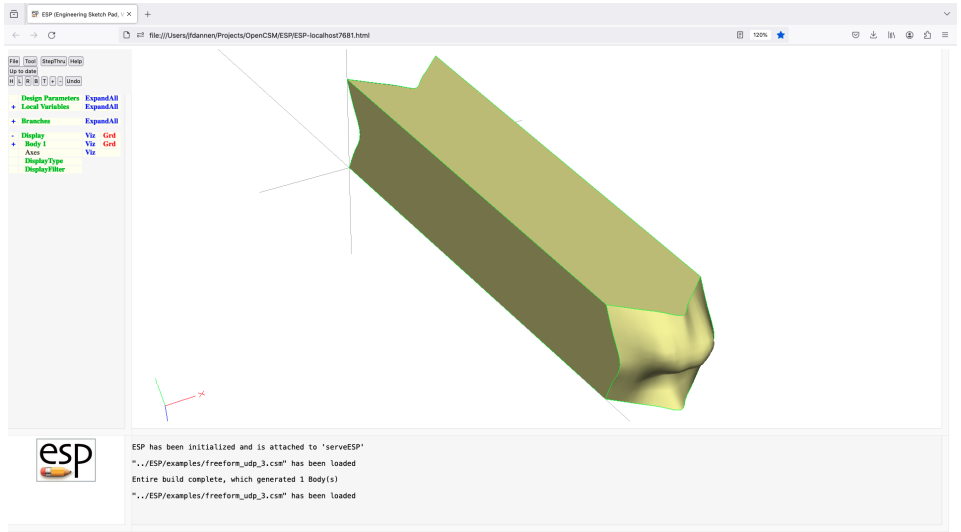




UDP `freeform` (2)

`$filename imax=1 jmax=1 kmax=1 xyz[]`

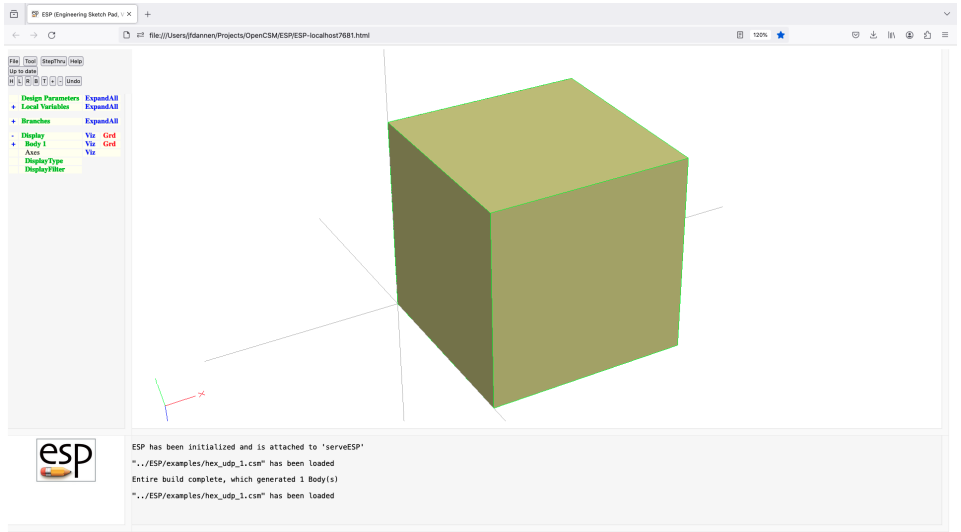
Create a freeform WireBody, SheetBody, or SolidBody from a file

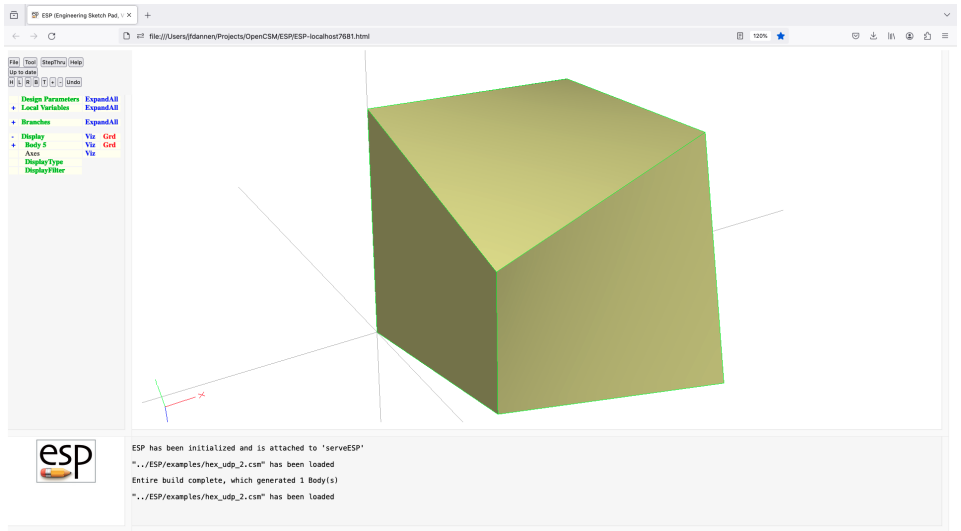


UDP `hex`

`corners[] uknots[] vknots[] wknots[] @@area @@volume`

Create general hexahedron SolidBody from its corners

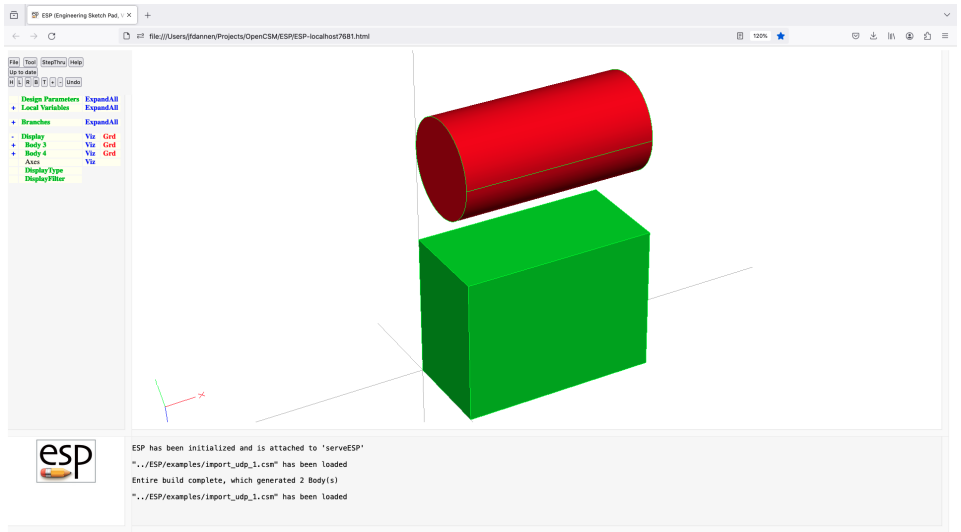


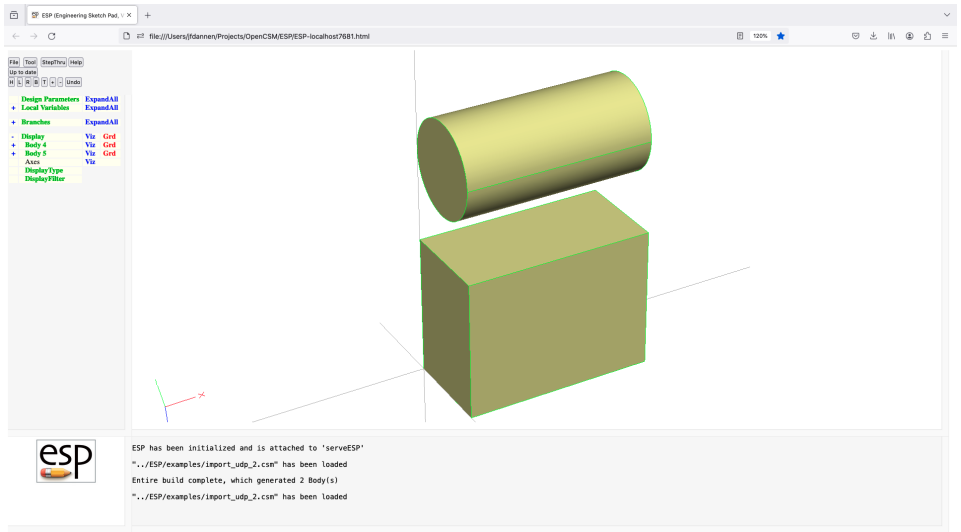


UDP import

$filename bodynumber=1 @@numbodies

Read a Body (or Bodys) from a file

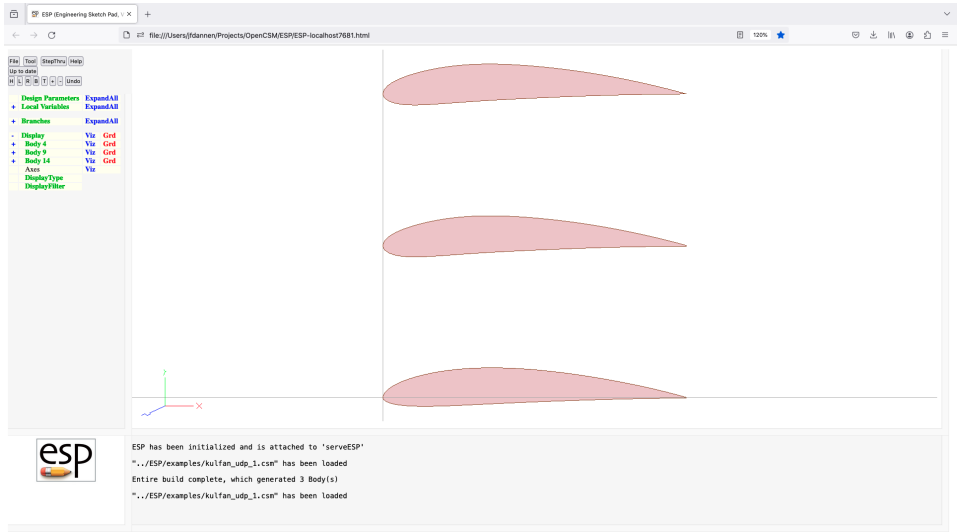




UDP `kulfan`

`class[] ztail[] aupper[] alower[]`

Create a Kulfan SheetBody airfoil



UDP naca456

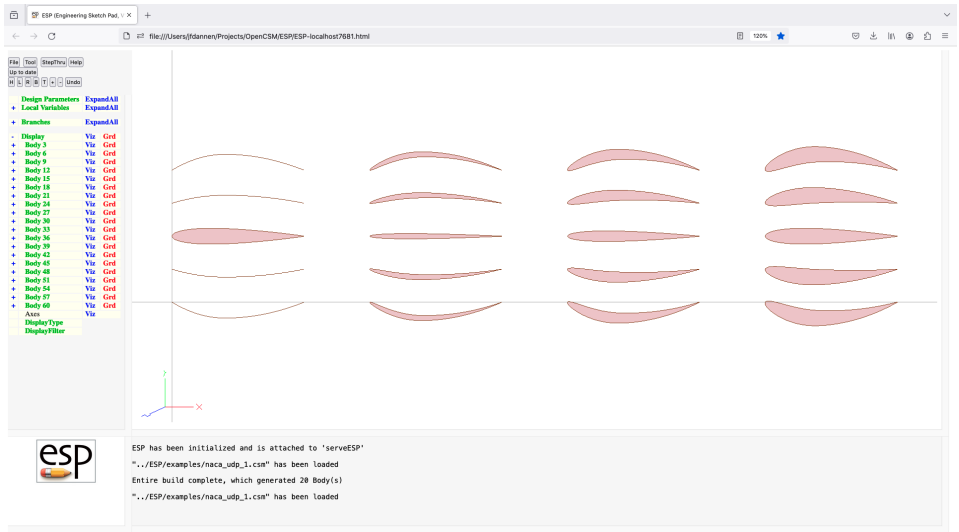
naca456 thkcode toc xmaxt leindex camcode cmax xmaxc cl a

Create a NACA 4-, 5-, or 6-series SheetBody airfoil

UDP naca

series=0012 thickness=0 camber=0 maxloc=0.4 offset=0 sharpte=0

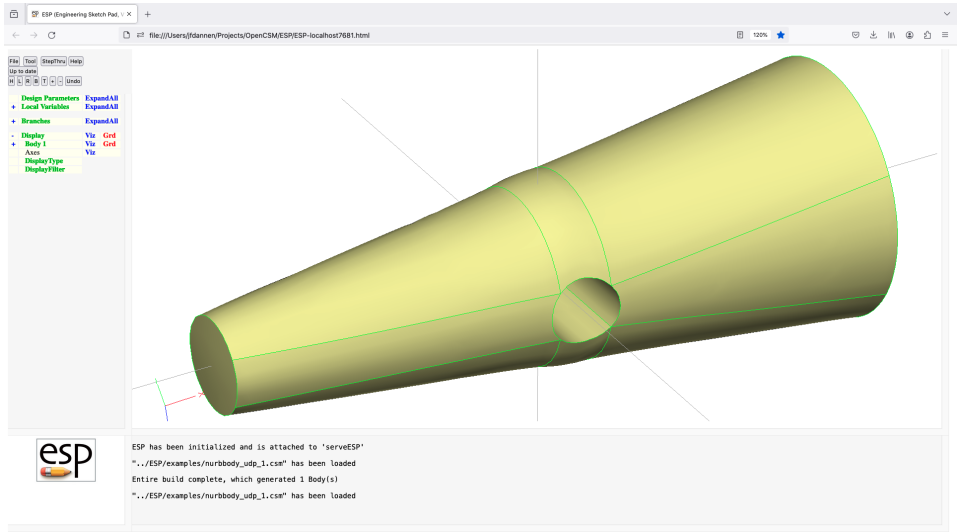
Create a NACA 4-series SheetBody airfoil or WireBody camberline



UDP nurbbody

$filename

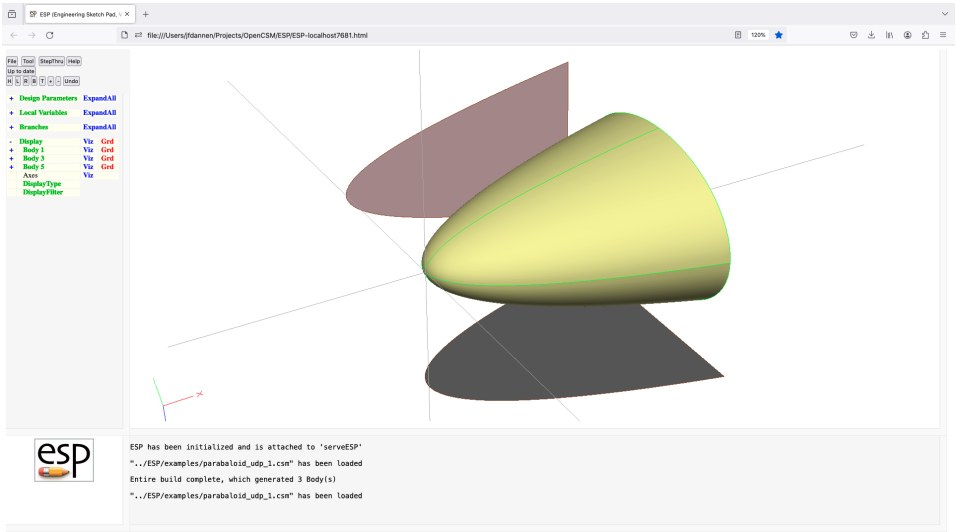
Create a Body from a series of NURBS



UDP parabaloid

xlength yradius=0 zradius=0

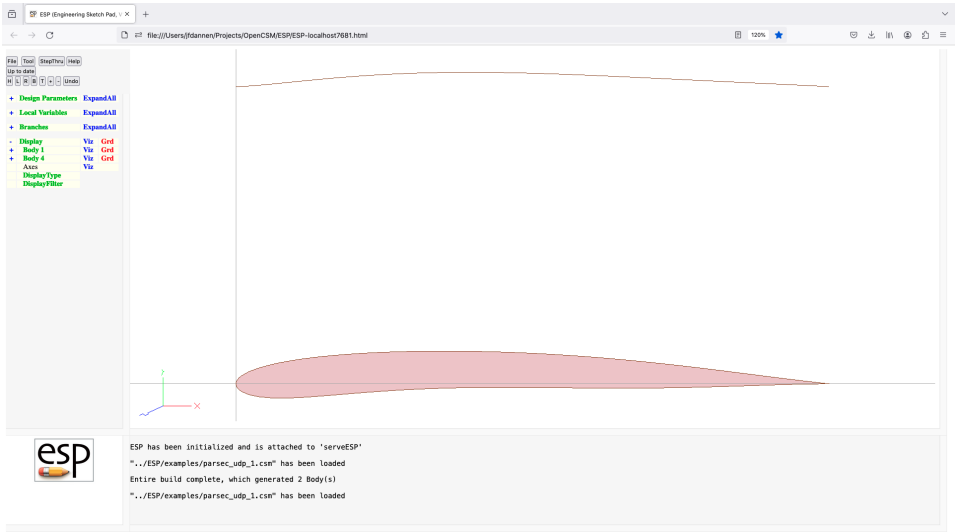
Create a parabaloid SolidBody or parabola SheetBody



UDP parsec

yte poly[] param[] meanline ztail[]

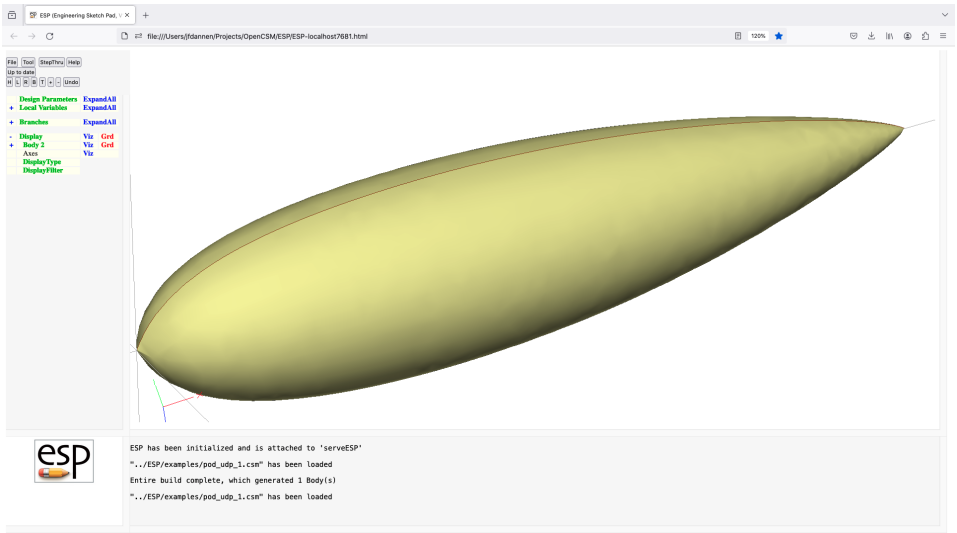
Create a Parsec SheetBody airfoil using either Sobieski's parameters or spline parameters



UDP pod

`length=0 fineness=0 @@volume`

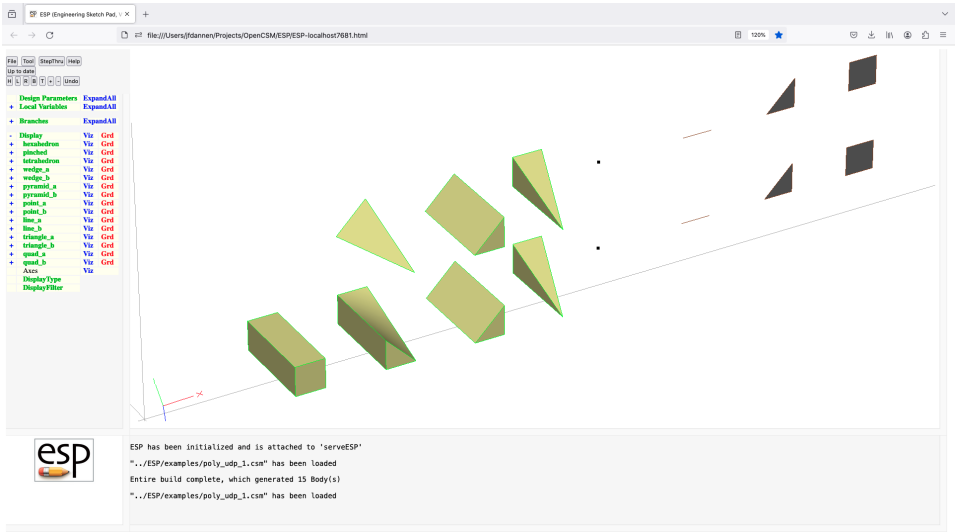
Create an `OpenVSP`-like SolidBody pod



UDP `poly`

`points[]`

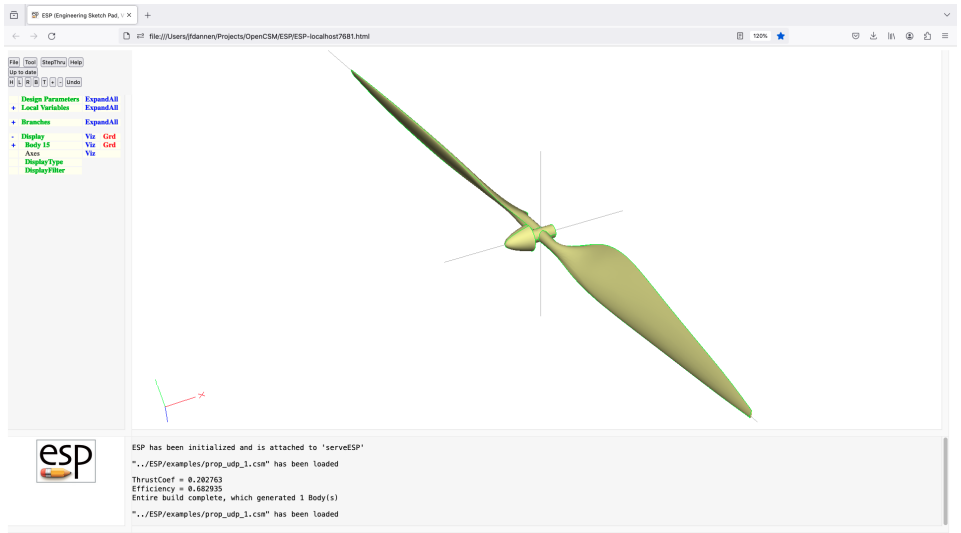
Create a general SolidBody polyhedron, SheetBody polygon, WireBody line, or NodeBody



UDP prop

```
prop nblade cpower lambda reyr rtip rhub clift cdrag alfa shdiam=0
shxmin shxmax spdiam=0 spxmin @@cthrust @@eff
```

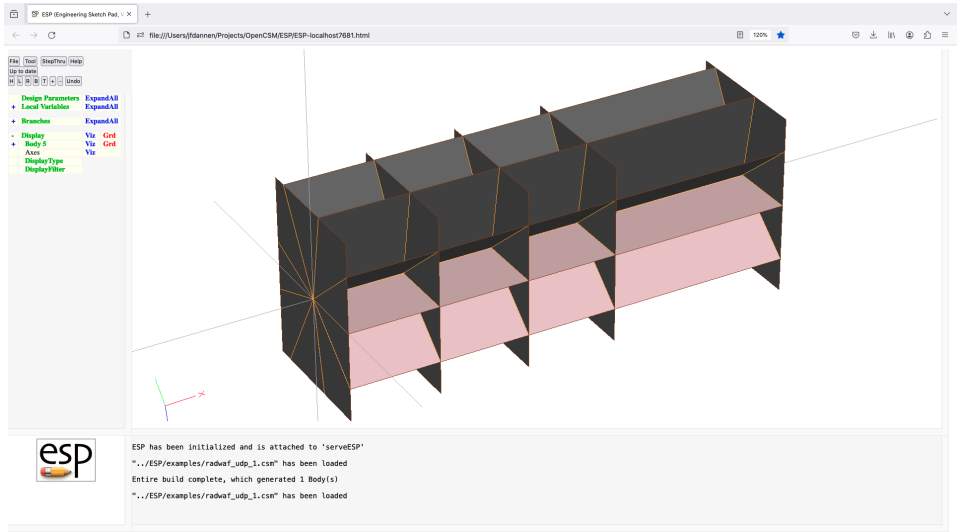
Create a propeller and optional shaft and spinner



UDP `radwaf`

`ysize=0 zsize=0 nspoke=0 xframe[]`

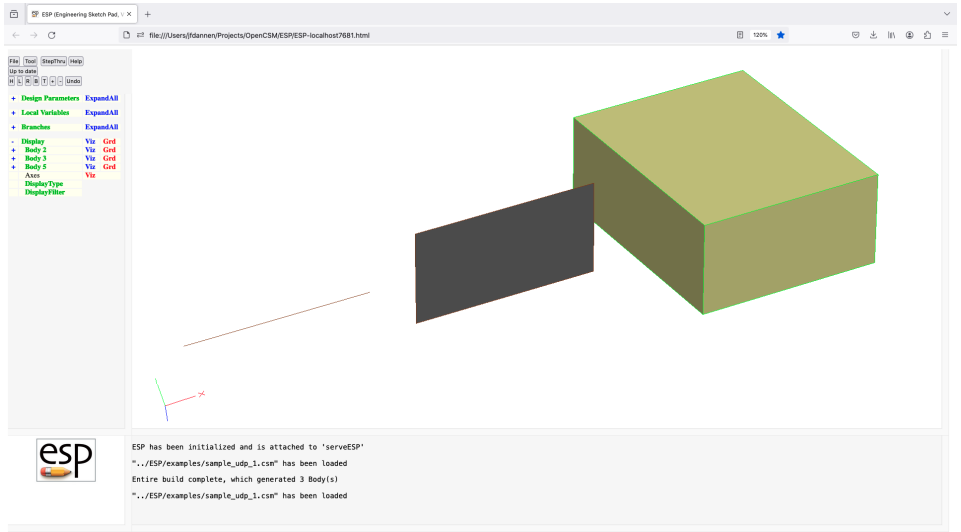
Create a radial SheetBody waffle



UDP sample

dx dy dz center[] @@area @@volume

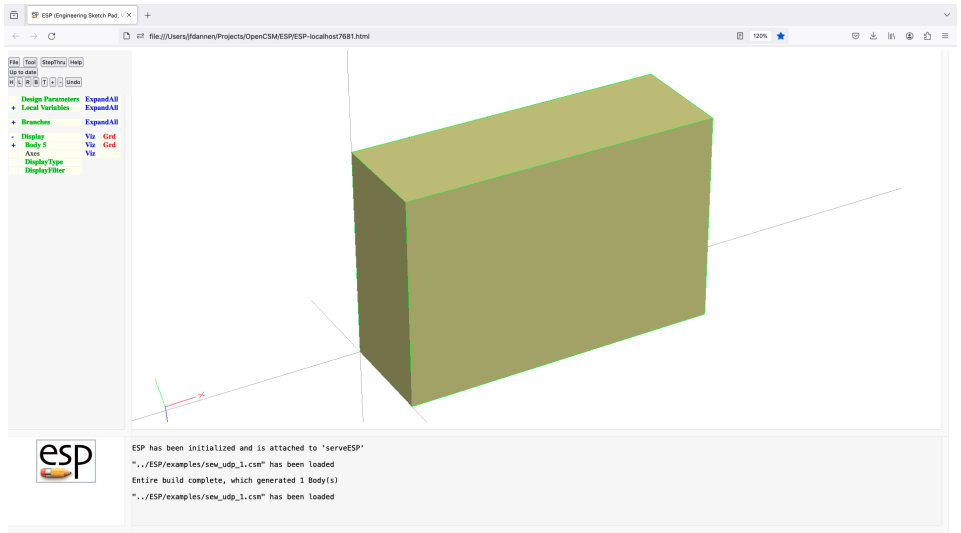
Used as an example for writing UDPs



UDP `sew`

`$filename toler=0 bodynum=1`

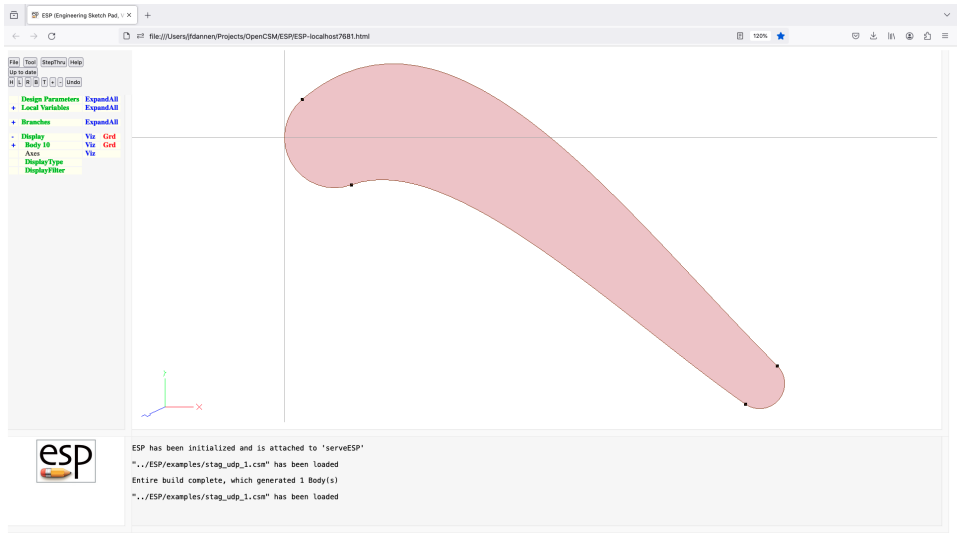
Sew Faces in a `step` file into a SolidBody



UDP stag

```
stag rad1=0.1 beta1=30 gama1=10 rad2=0.05 beta2=-40 gama2=5
alfa=-30 xfrnt=0.333 xrear=0.667
```

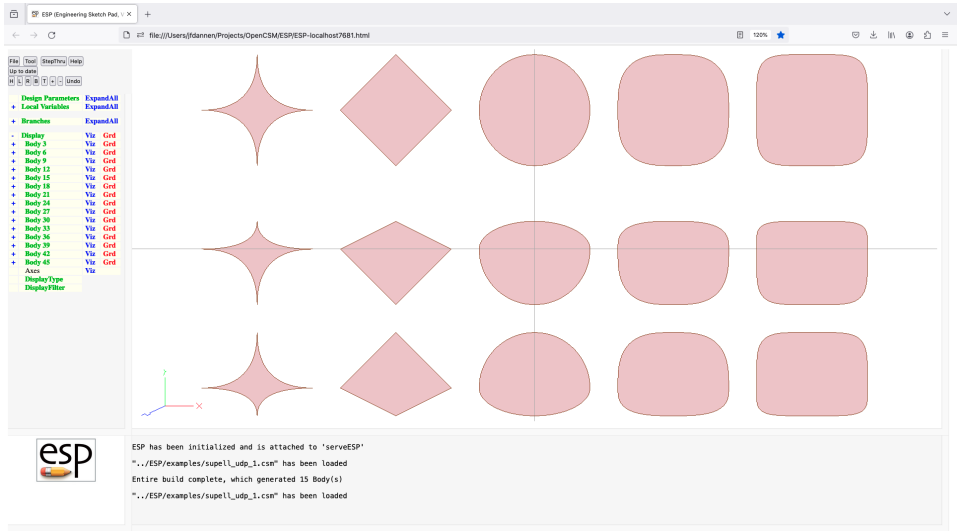
Create a simple turbomachinery SheetBody airfoil

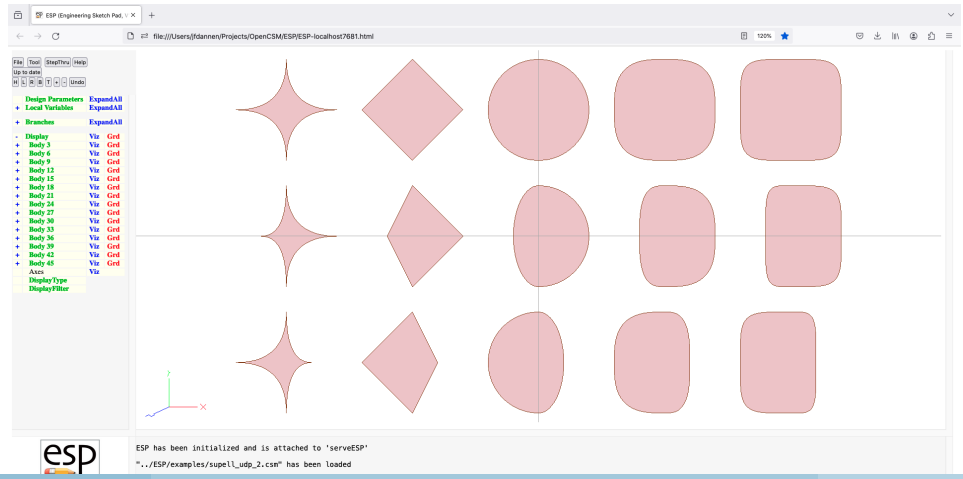


UDP `supell` (1)

`rx rx_w rx_e ry ry_s ry_n n n_w n_e n_s n_n n_sw n_se n_nw n_ne offset nquad=4`

Create a super-ellipse SheetBody or WireBody

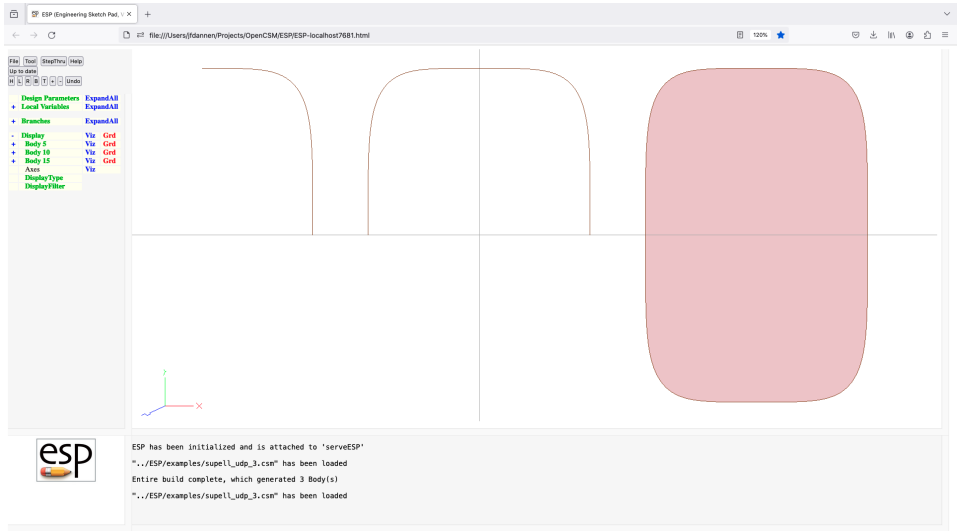


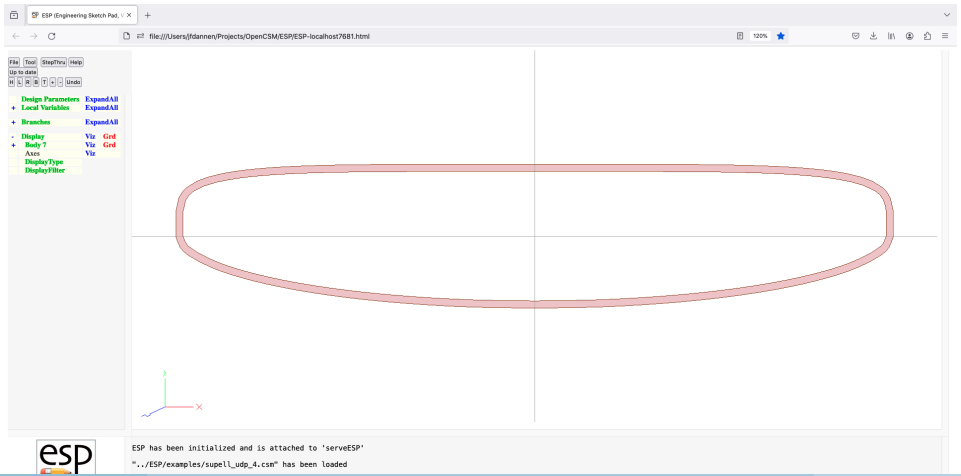


UDP supell (2)

`rx rx_w rx_e ry ry_s ry_n n n_w n_e n_s n_n n_sw n_se n_nw n_ne offset nquad=4`

Create a super-ellipse SheetBody or WireBody

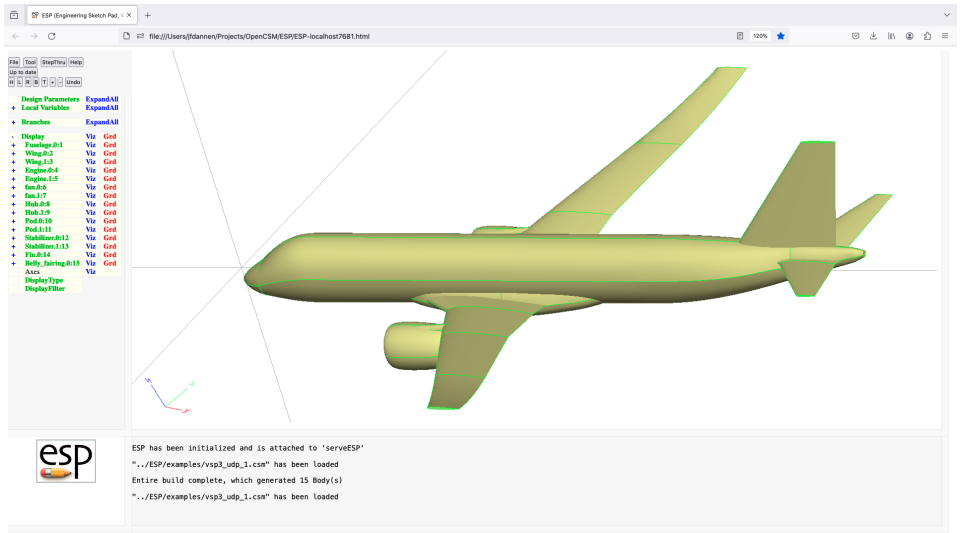




UDP vsp3

$filename keeptemps=0

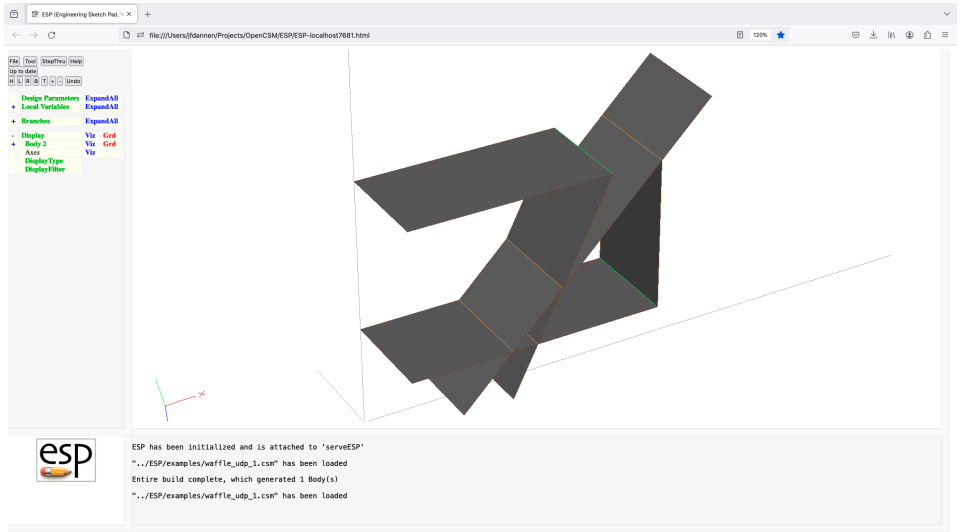
Create SolidBodys or SheetBodys from an OpenVSP file

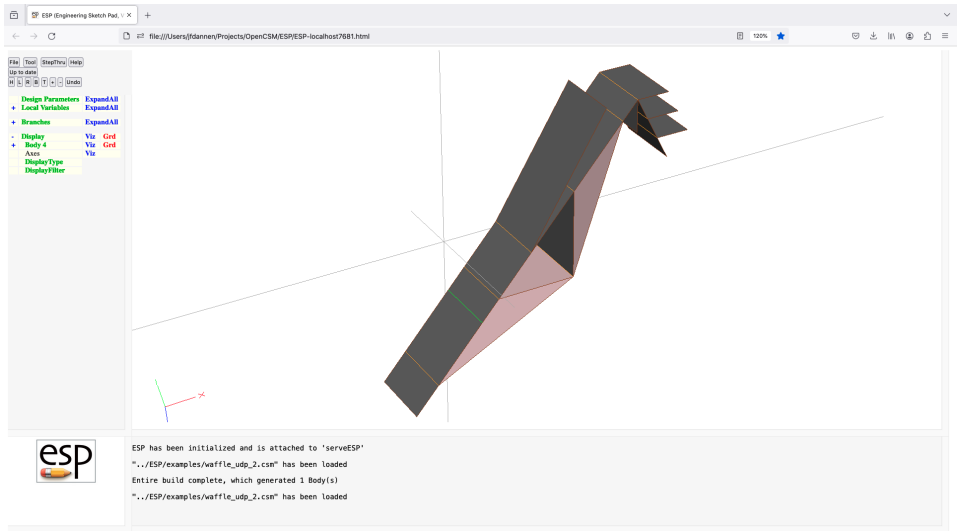


UDP `waffle`

`depth=1 segments[] $filename progress=0`

Create a SheetBody waffle by extruding a 2D group of segments

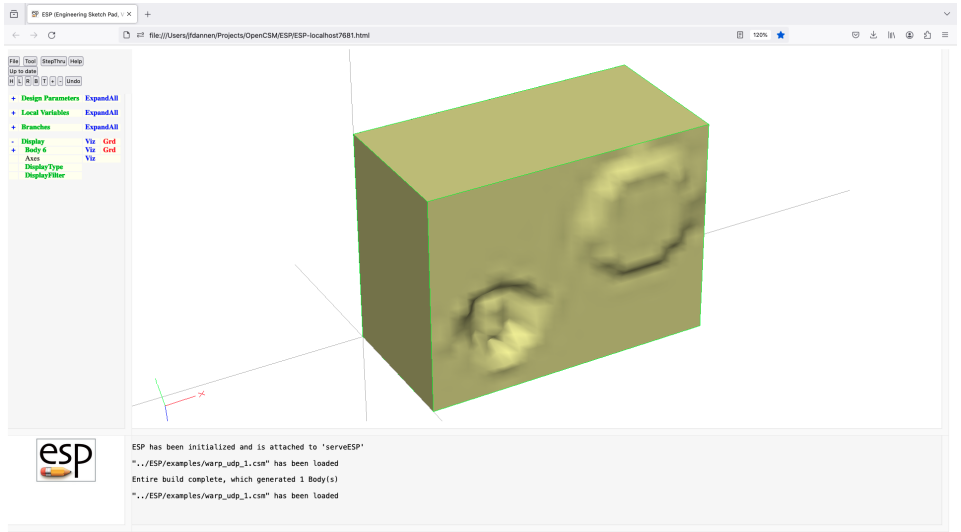




UDP `warp`

`$egadeFile iface dist[] toler=0`

Warp one Face of a Body via control point movement



Shipped User-defined Functions

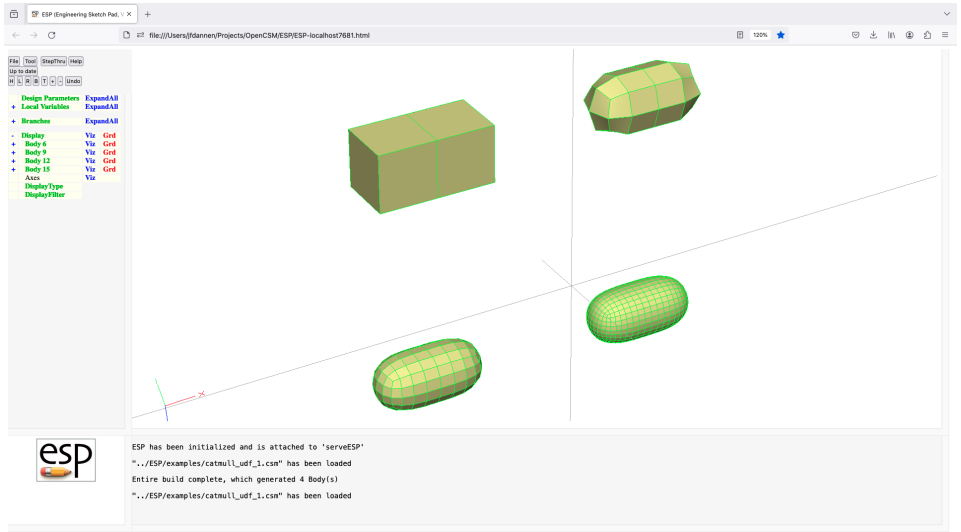
- catmull
- compare
- createBEM
- createPoly
- deform
- droop
- dumpPmtrs
- editAttr
- flend
- ganged
- guide
- linalg
- matchBodys
- mechanism
- naca6mv
- nacelle
- nuscale
- offset
- printBbox
- printBrep
- printEgo
- shadow
- slices
- stiffener

- UDFs are called in exactly the same way as UDPs are called

UDF catmull

nsubdiv=1 progress=0 @@area @@volume

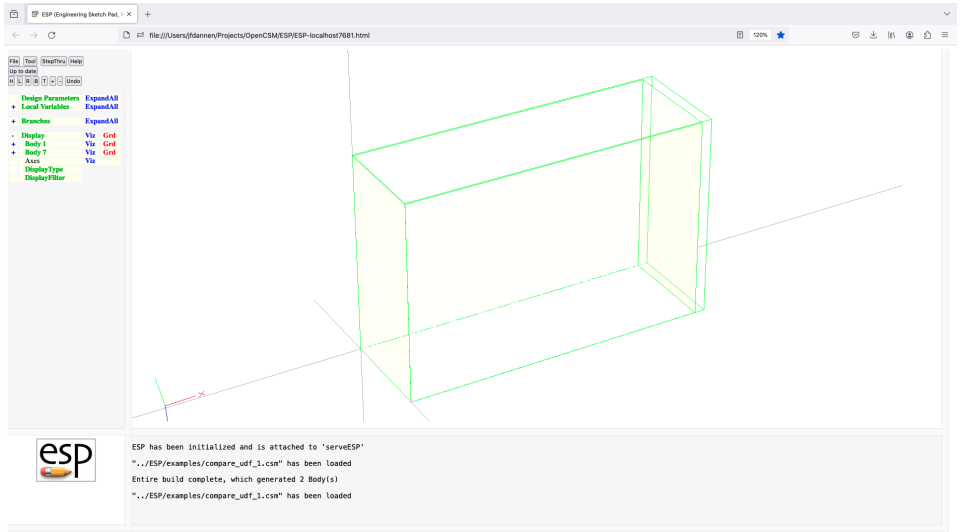
Create Catmull-Clark subdivision surfaces from Body on Stack



UDF compare

$tessfile $histfile $plotfile toler=1e-6

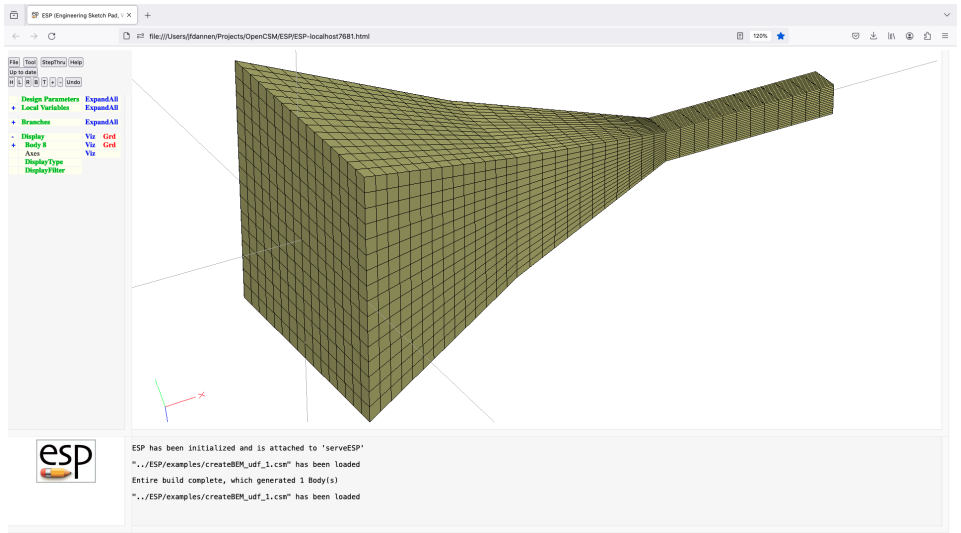
Compare points in `tessfile` and Body on Stack



UDF createBEM

```
$filename space=0 imin=3 imax=5 nocrod=0
```

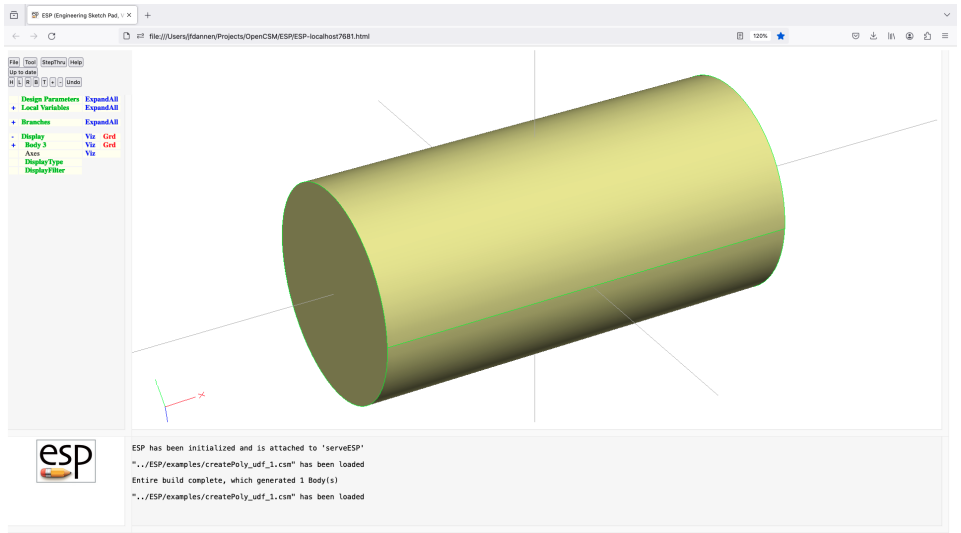
Create a NASTRAN-type built-up element (BEM) file from Body on Stack



UDF createPoly

$filename hole[]

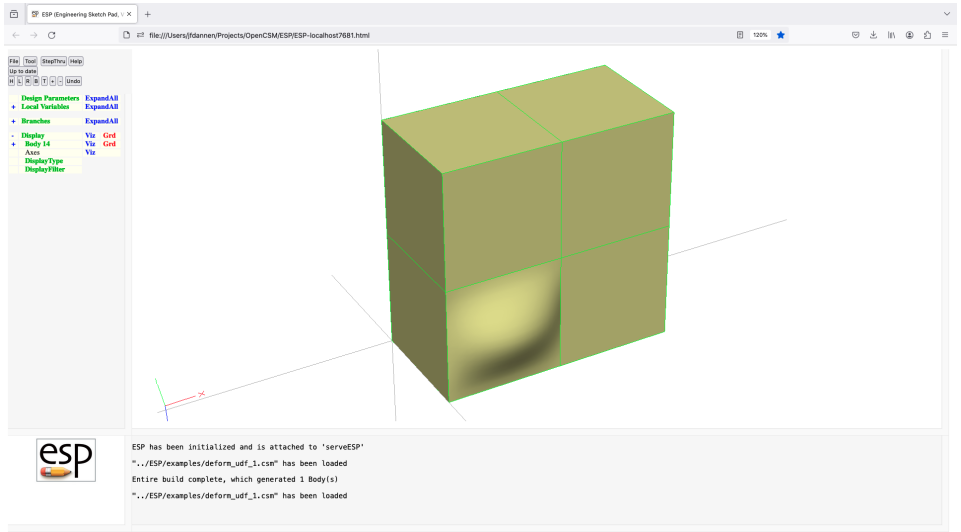
Create a TETGEN .poly file for volume between 2 Bodys on Stack

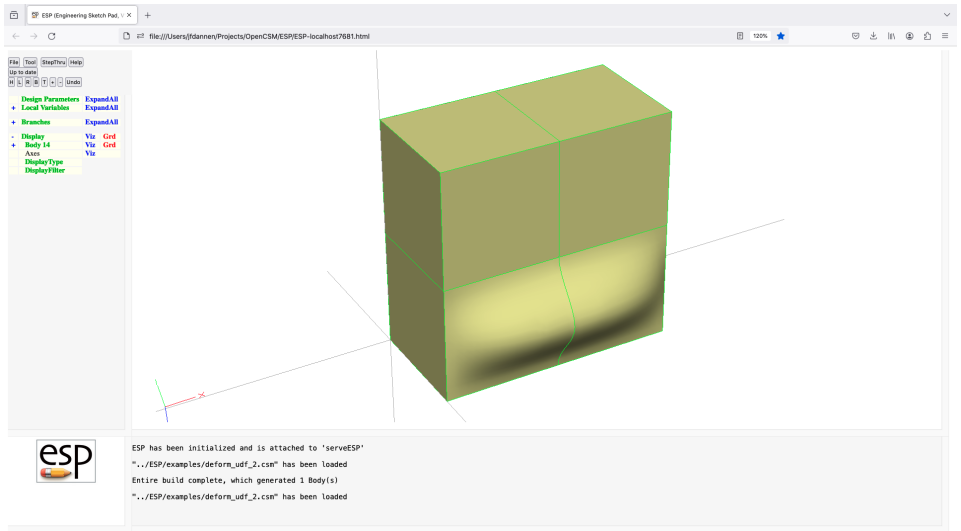


UDF deform

iface[] iu[] iv[] dist[] toler=0

Deform Bsplines on Body on Stack





UDF droop

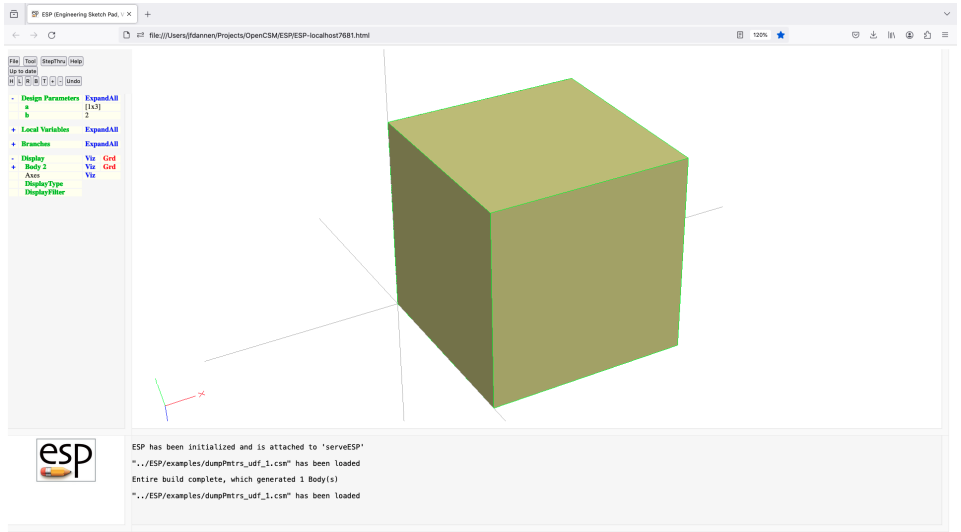
`xle=-100 thetale=0 xte=100 thetate=0`

Applied leading- and/or trailing-edge droop to SheetBody on Stack

UDF dumpPmtrs

$filename

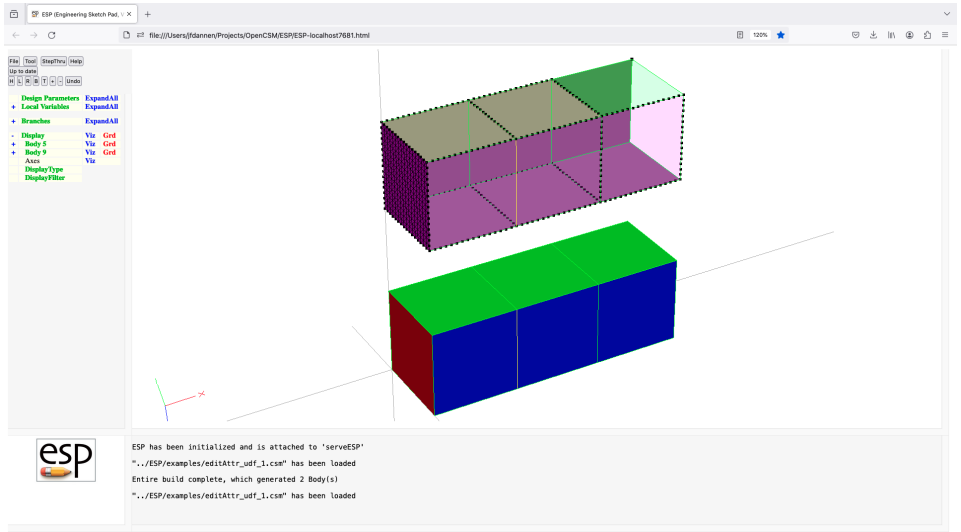
Dump all Parameters to a file as a json dict

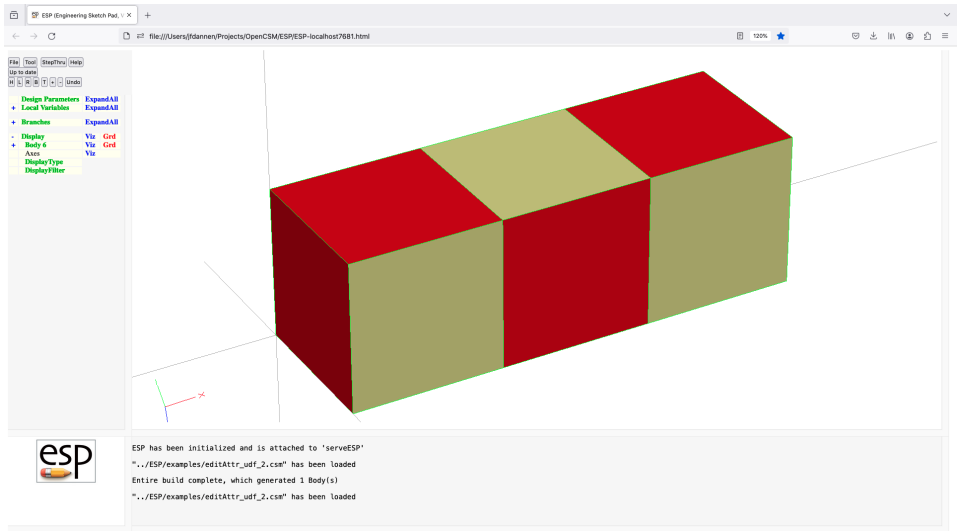


UDF editAttr

$attrname $input $output overwrite=0 $filename verbose=0 @@nchange

Edits Attributes for Body on Stack

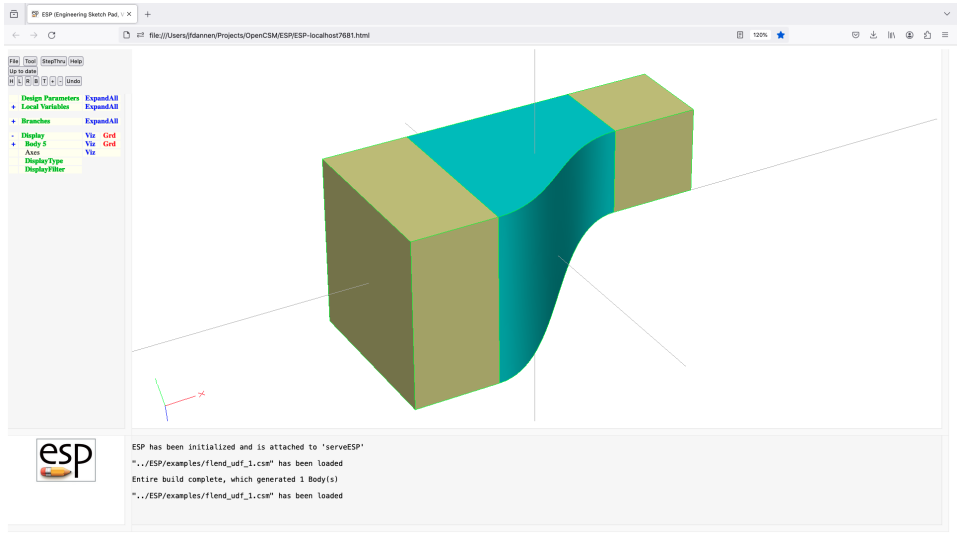


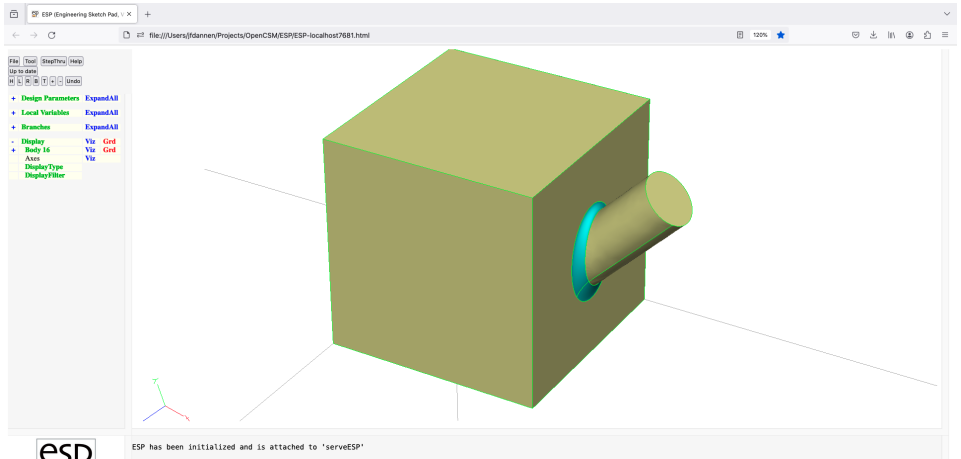


UDF `flend` (1)

`slopea=1 slopeb=1 toler=1e-6 equis=0 npnt=33 plot=0`

Create a flend (similar to fillet) that connects the one or two Bodys on Stack

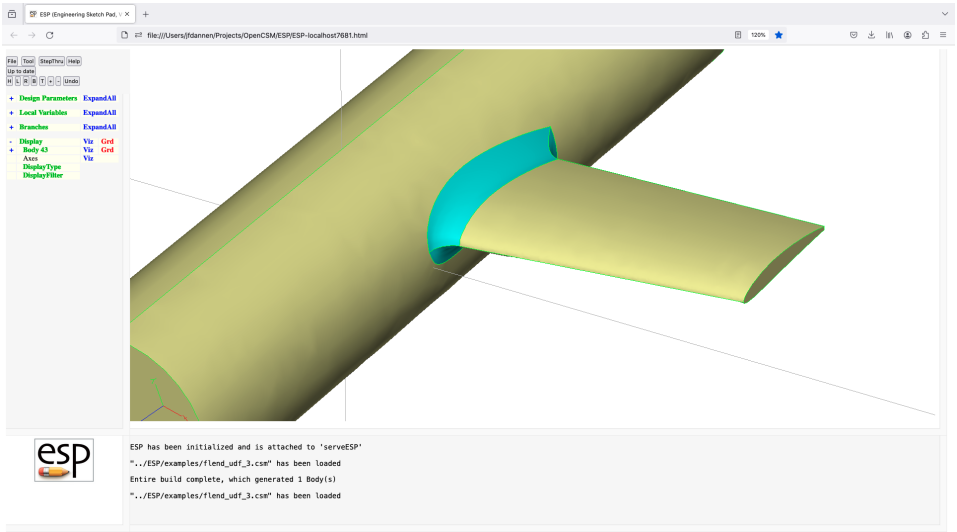




UDF `flend` (2)

`slopea=1 slopeb=1 toler=1e-6 equis=0 npnt=33 plot=0`

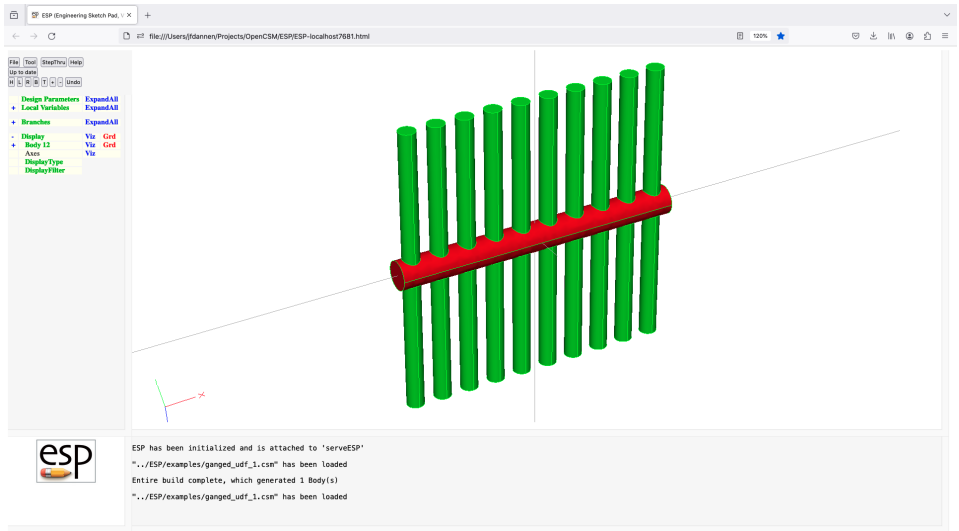
Create a flend (similar to fillet) that connects the one or two Bodys on Stack

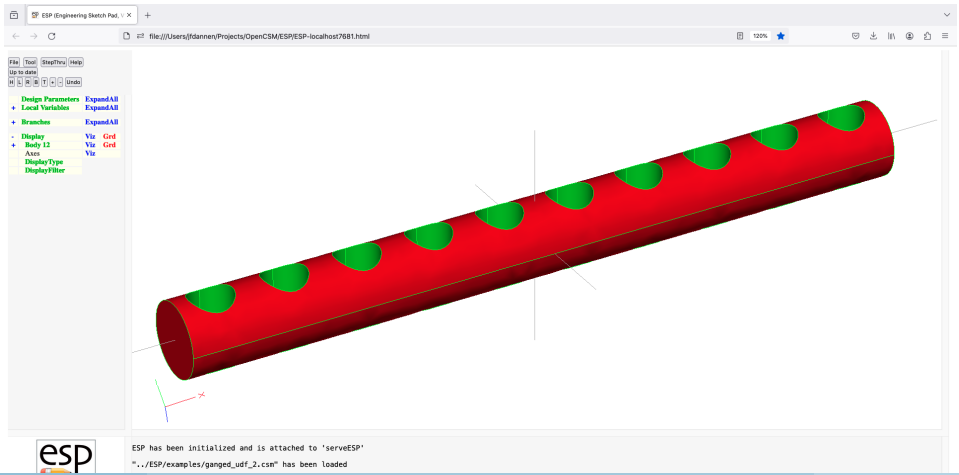


UDF ganged (1)

`$op toler=0`

Performed ganged SUBTRACTs or UNIONs to Bodys on Stack back to Mark

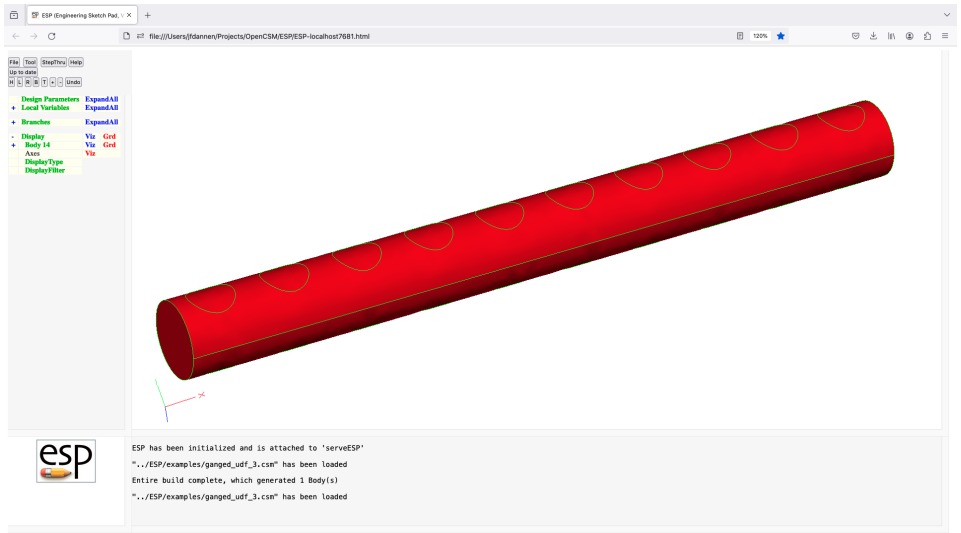




UDF ganged (2)

`$op toler=0`

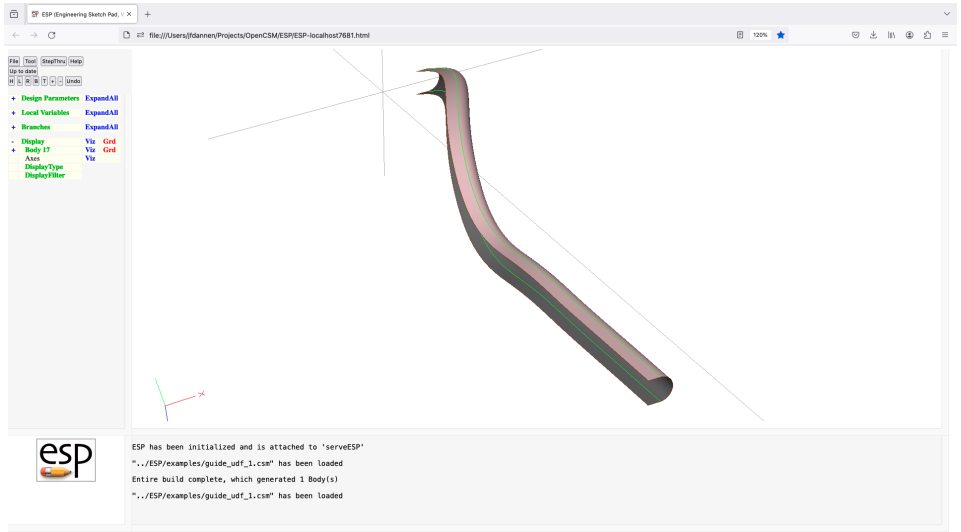
Performed ganged SUBTRACTs or UNIONs to Bodys in Stack back to Mark

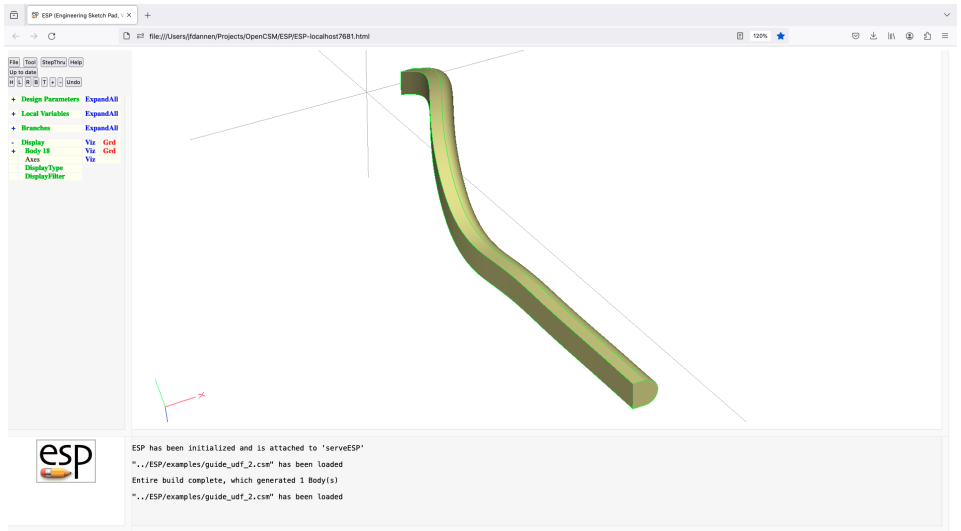


UDF guide

nxsect=5 origin=0 axis=0

Sweep a SheetBody or WireBody along a WireBody guide curve

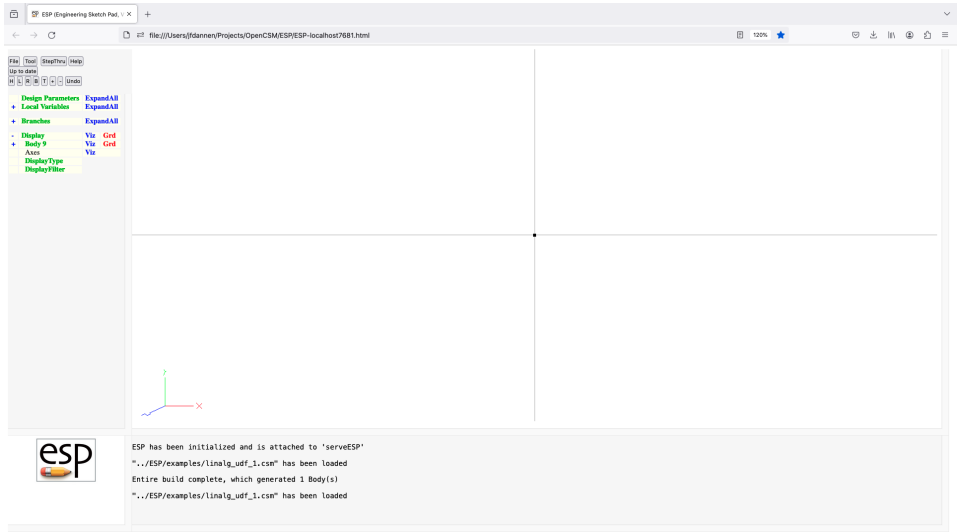




UDF `linalg`

`$oper m1[] m2[] @@ans[]`

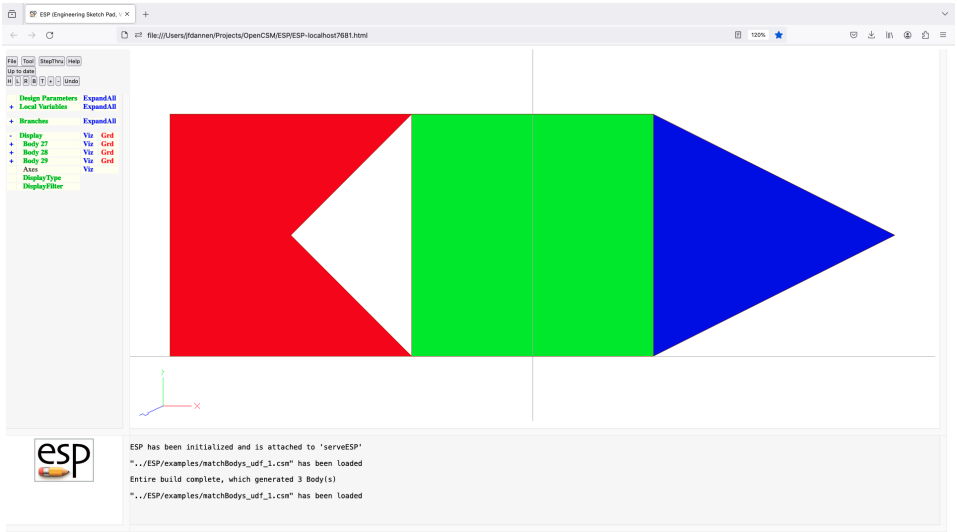
Perform linear algebra operations



UDF matchBodys

toler @@nnodes @@nedges @@nfaces

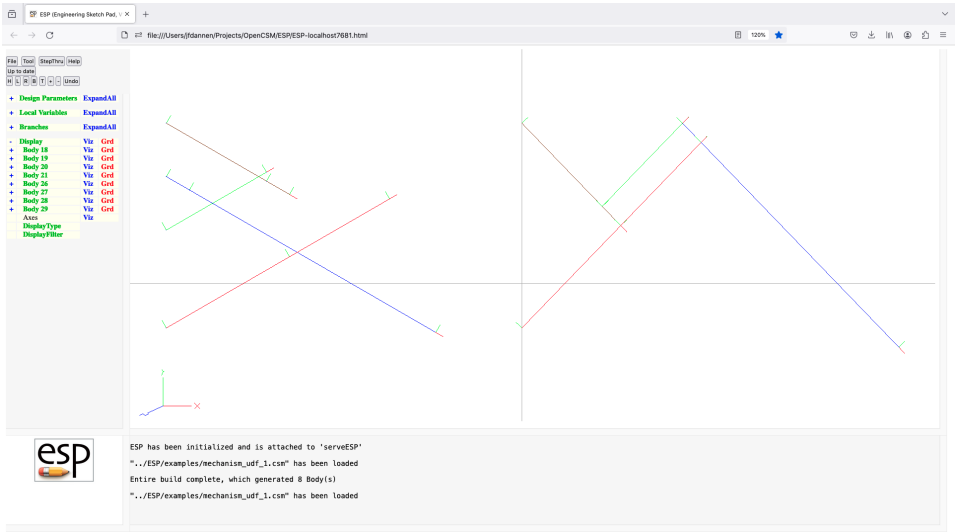
Returns number of Nodes, Edges, and Faces that match for two Bodys on the Stack



UDF `mechanism`

`$fixed[]`

Solves mechanism equations and moves Bodys



UDF naca6mc

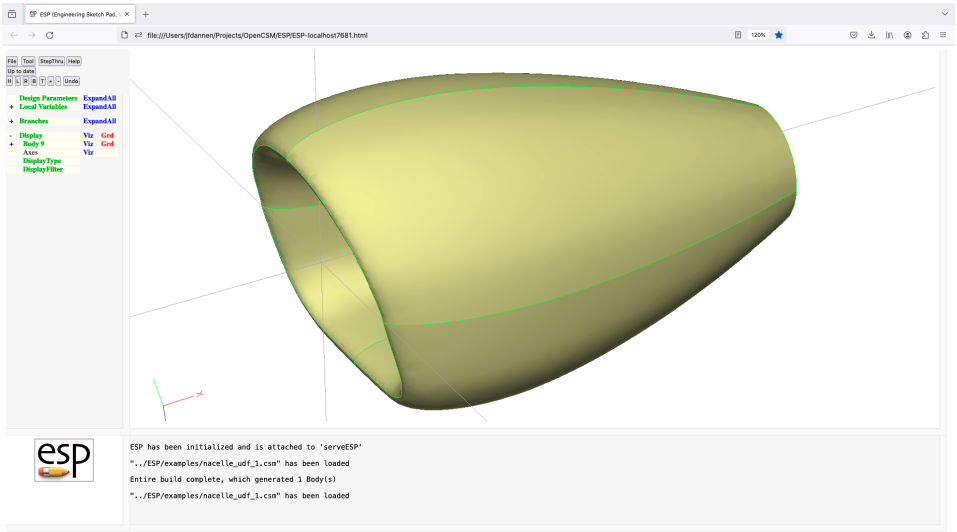
clt[] a[]

Modifies SheetBody on Stack with a group of 6-series NACA camber distributions

UDF nacelle

f_rad a_rad f_pow a_pow length deltah rakeang

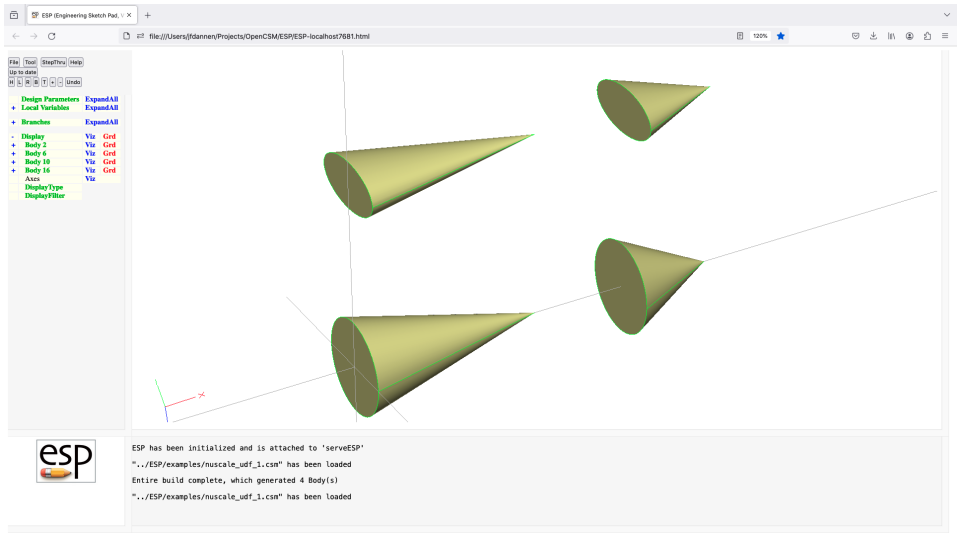
Create a aircraft engine nacelle SolidBody from the SheetBody profile on the Stack



UDF nuscale

xscale=1 yscale=1 zscale=1 xcent=0 ycent=0 zcent=0

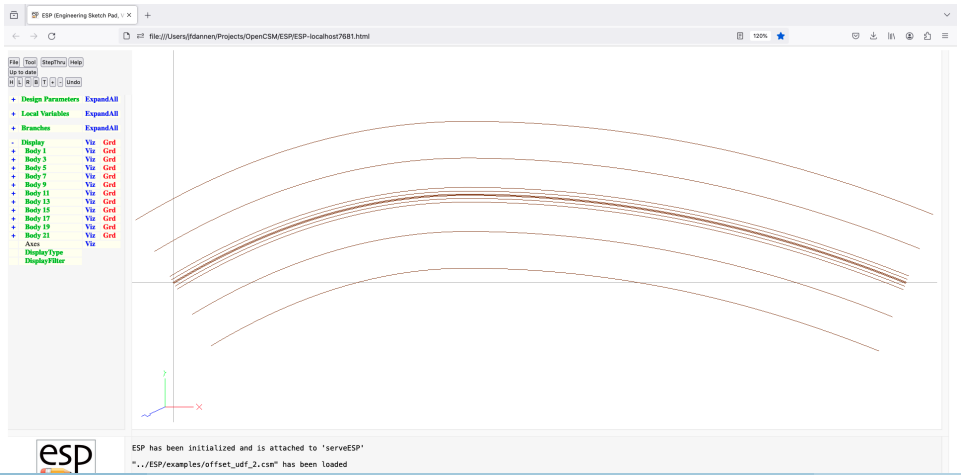
Convert Body on Stack to Bsplnes and applies scaling in each coordinate direction



UDF offset (1)

nodelist[] nodedist[] edgelist[] facelist[] dist

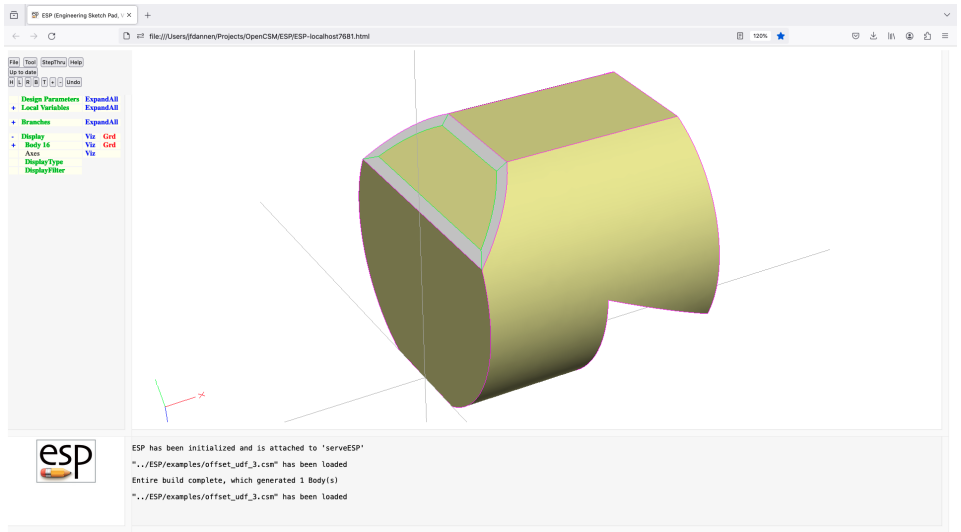
Create offset WireBodys or scribes the Body on Stack with offset curves

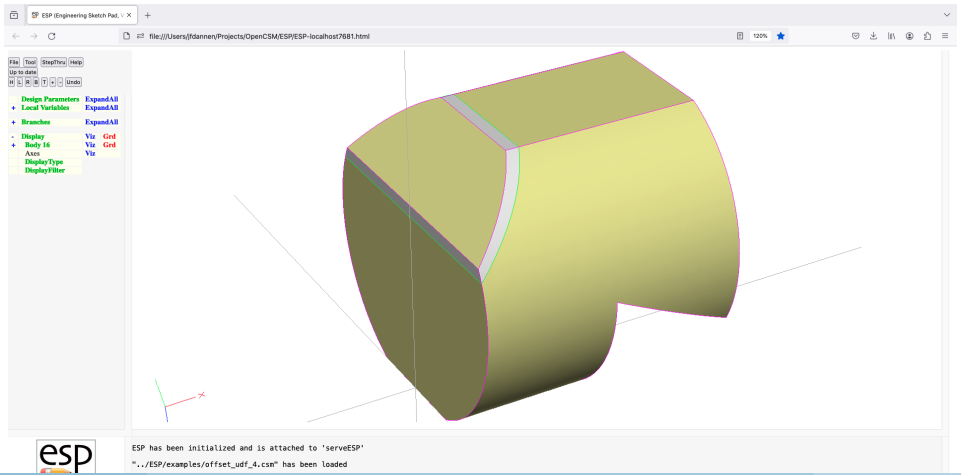


UDF offset (2)

nodelist[] nodedist[] edgelist[] facelist[] dist

Create offset WireBodys or scribes the Body on Stack with offset curves

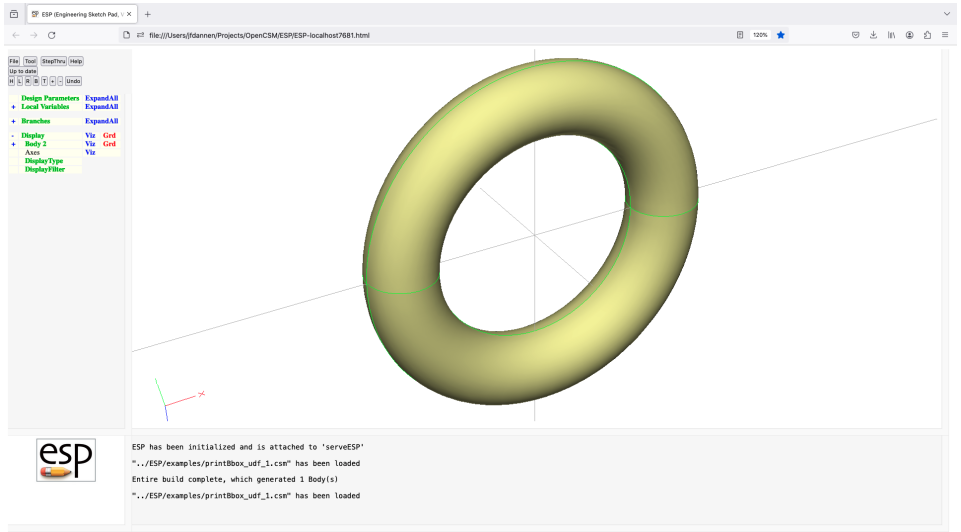




UDF printBbox

-

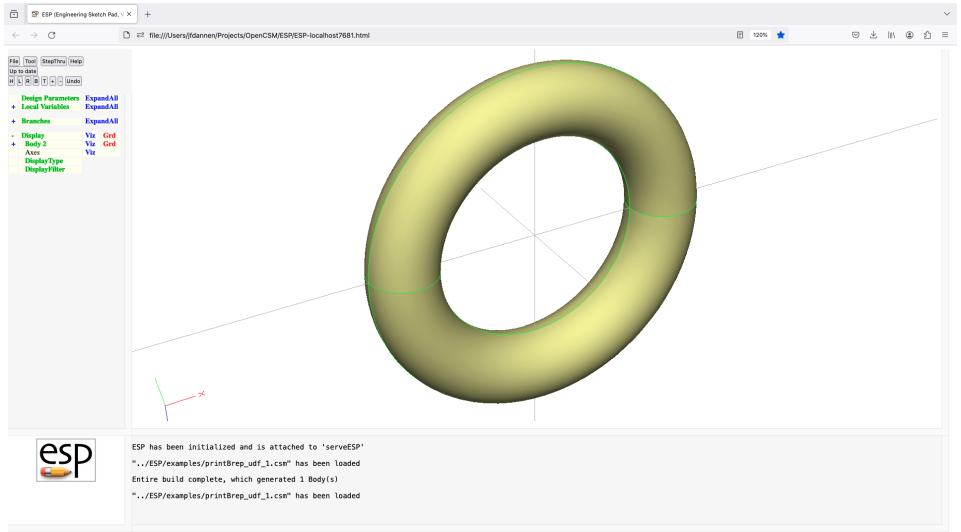
Print bounding box of Body on Stack



UDF printBrep

-

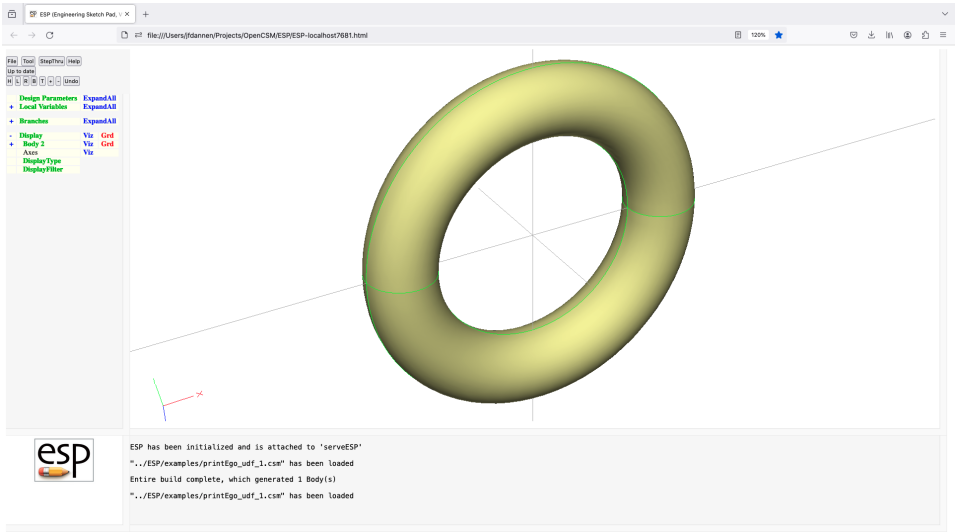
Print boundary representation of Body on Stack



UDF printEgo

-

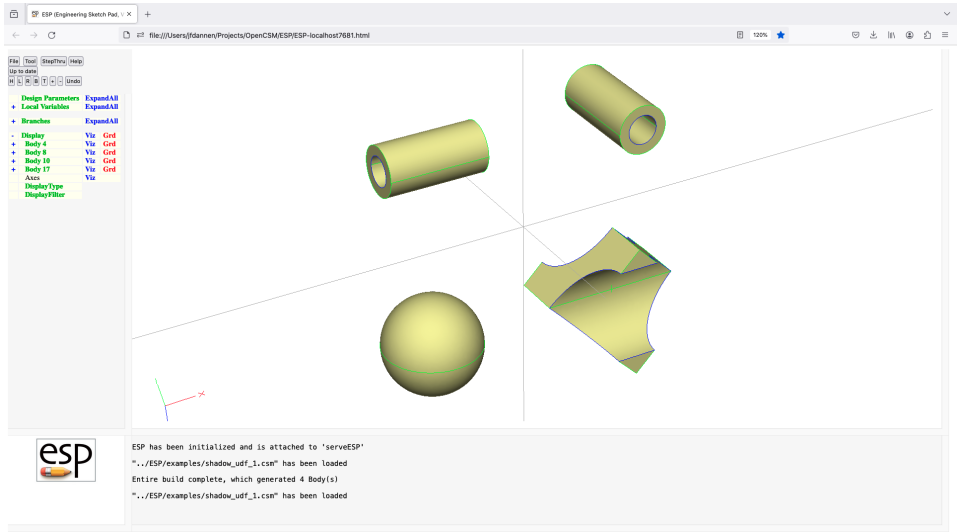
Print EGADS ego of Body on Stack



UDF shadow

numpts=1001 @@area @@xcent @@ycent @@ixx @@iyy

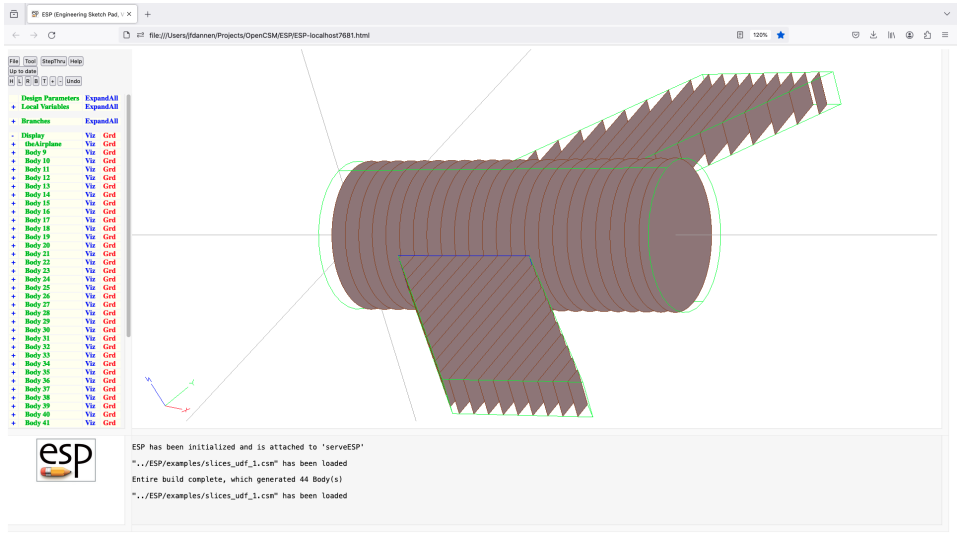
Find mass properties of Z-direction-shadow of Body on Stack

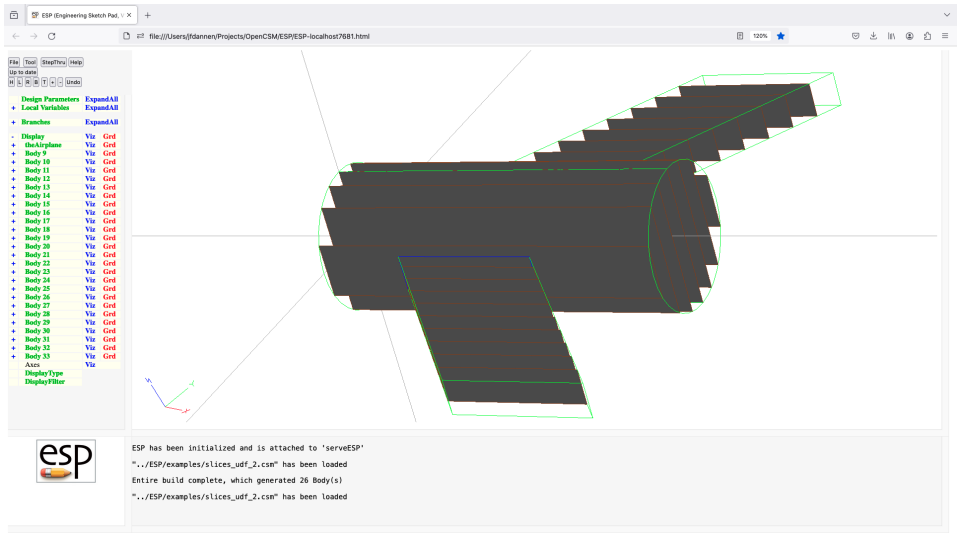


UDF slices (1)

nslice $dirn

Create SheetBody slices of Body on Stack

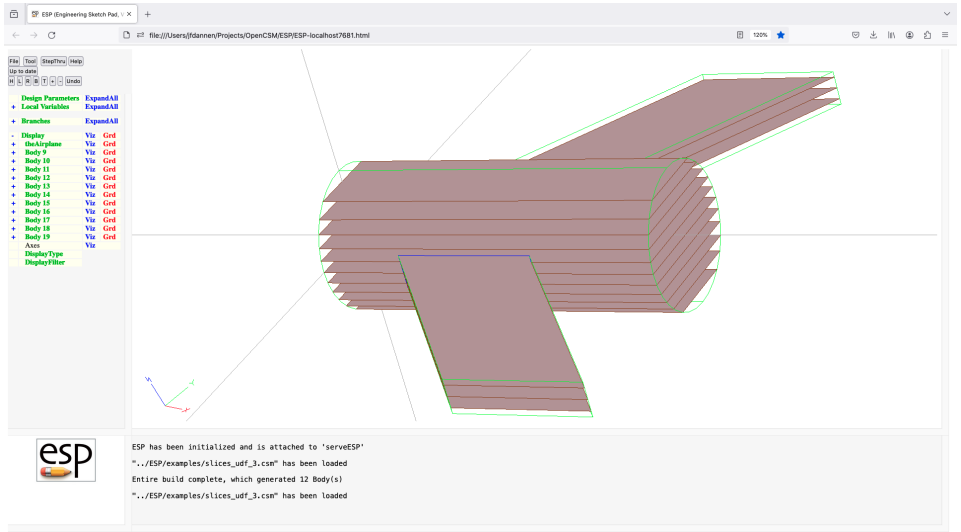




UDF slices (2)

nslice $dirn

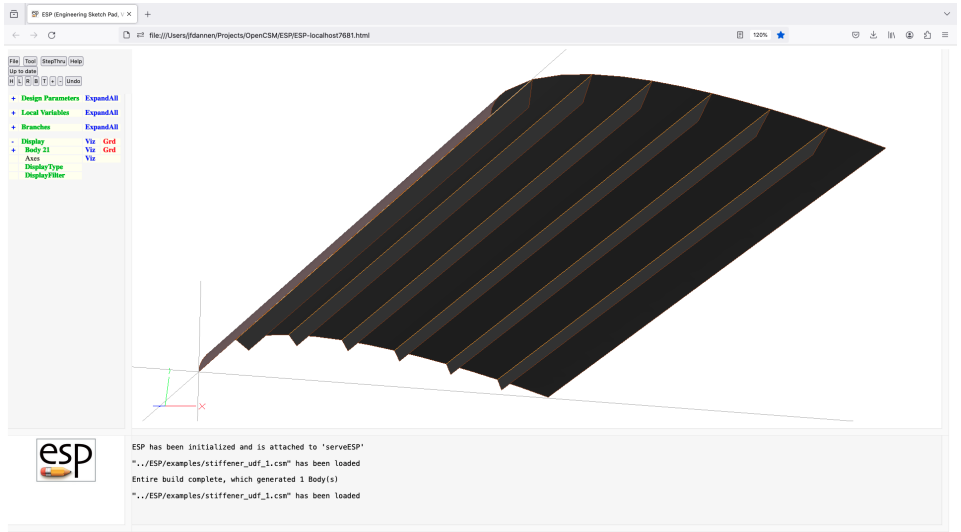
Create SheetBody slices of Body on Stack



UDF `stiffener`

`stiffener beg[] end[] depth=0 angle=0`

Adds stiffener that is orthogonal to SheetBody on stack



Writing Your Own UDP or UDF

- See `EngSketchPad/doc/UDP_UDF/udp_udf.pdf`

- Demonstrates bottom-up build
  - Node→Edge→Loop→Face→Shell→Body
- Demonstrates top-down build
  - Boolean operations
  - appied features
  - ...
- Written in C

User-defined Components (UDCs)

- A UDC is a series of statements that are contained in a `.udc` file
- The statements in the UDC can be treated in either one of two ways:
  - Include-style
    - statements within the UDC are simply processed as if they were included in the enclosing `.csm` or `.udc` file
    - the `.udc` file must start with an `INTERFACE . ALL` statement
    - Variables and Parameters in the `.udc` file have the same scope as its caller (that is, the UDC shares variables with its caller)
  - Function-style
    - Variables and Parameters in the `.udc` file have local scope (that is, the UDC's variables are private)
    - Variables in the UDC get values via `INTERFACE . IN` statements
    - The UDC can output some of its variables via `INTERFACE . OUT` statements

Top-level Scope

- The `.csm` (or `.cpc`) file is at top-level scope
- Any include-style UDC whose caller has top-level scope also has top-level scope

Calling a UDC

- UDCs are called with a `UDPRIM` statement
- `$primtype` must start with a slash (`/`), dollar-slash (`$/`), or dollar-dollar-slash (`$$/`)
  - if `/`, then the UDC file is in the current working directory
  - if `$/`, then the UDC file is in the same directory as the `.csm` file
  - if `$$/`, then the UDC file is in `ESP_ROOT/udc` directory
  - if `$$$/`, then the UDC file is in `ESP_UDC_PATH` directory
- The `UDPRIM` statement can be preceded by one or more `UDPARG` statements
- name-value pairs are processed in order (with possible over-writing)

Example Include-style UDC

- In `test1.csm`

```
SET     A    1
SET     B    10
SET     C    0
UDPRIM $/test2
SET     D    C^2
```

- In `test2.udc`

```
INTERFACE   .   ALL
SET             C   A+B
```

- After running, `C`=11 and `D`=121

Example Function-style UDC

- In `test3.csm`

```
SET     A    1
SET     B    10
SET     C    0
UDPRIM $/test4  first A  second B
SET     D    C^2
```

- In `test4.udc`

```
INTERFACE  first   IN  0
INTERFACE  second  IN  0
INTERFACE  sum     OUT 0
SET        C       999
SET        sum     first+second
```

- After running, `C`=0, `D`=0, and `@@sum`=11

UDCs Shipped with ESP (1)

- `applyTparams factor=1`
  - apply `.tParams` to the Edges and Faces of the Body on the top of the Stack
- `biconvex thick=0`
  - generate a biconvex airfoil
- `boxudc dx=0 dy=0 dz=0 @@vol`
  - similar to the box UDP
- `contains @@contains`
  - determine if either of the two Bodys on the top of the Stack contains the other
- `diamond thick=0`
  - generate a double-diamond airfoil
- `duct diameter=1 length=2 thickness=0.10 camber=0.04`
  - generate a duct

UDCs Shipped with ESP (2)

- `expressions xx yy zz @@aa @@bb`
  - a test UDC that has no other practical use
- `flapz xflap[] yflap[] theta=15 gap=0.01 openEnd=0`
  - cut a (deflected) flap in a Body
- `fuselage xloc zloc width height noselist taillist`
  - generate a fuselage
- `gen_rot xbeg=0 ybeg=0 zbeg=0 xend=1 yend=1 zend=1 rotang=0 @@azimuth @@elevation`
  - general rotation with two fixed points
- `overlaps @@overlaps`
  - determine if the two Bodys on the top of the Stack overlap each other
- `popupz xbx ybax height=1`
  - pop up a part of the configuration

UDCs Shipped with ESP (3)

- `spoilerz xbox ybox depth=1 thick=0.1 theta=30 overlap=0.002 extend=0.20`
  - pop up a spoiler
- `strut length=2.0 thickness=0.2 height=1.0 sweep=0`
  - generate a strut (between a duct and wing)
- `swap`
  - swaps the two Bodys or Marks on the top of the stack
- `wake mirror=0area=100 aspect=8 taper=0.8 twist=-5 sweep=0 dihedral=0 camber=0.04 wakeLen=3.0 wakeAng=0`
  - generate a wake
- `wing mirror=0 area=100 aspect=8 taper=0.8 twist=-5 sweep=0 dihedral=0 thickness=0.12 sharpte=0 camber=0.04 inboard=0 outboard=1 pctchord=0 angleft=0 angrite=0 spar1=0 spar2=0 nrib=0 @@span`
  - generate a wing

Writing a UDC

- Determine if you want include-style or function-style
- If function-style, define the interface
  - input variables (with default values)
  - output variables (with default values)
- Add assertions to ensure valid inputs
- Make sure all "output" variables are assigned values

Example UDC — swap.udc

```
# make sure that there are at least entities on the Stack
IFTHEN @stack.size LT 2
   THROW  999  # not enough entries on Stack
# if Mark,Mark on top of Stack
ELSEIF @stack[@stack.size-1] EQ 0  AND  @stack[@stack.size] EQ 0
# if Body,Mark on top of Stack
ELSEIF @stack[@stack.size] EQ 0
   STORE   .
   STORE   tempSwap 99
   MARK
   RESTORE tempSwap 99
# if Mark,Body on top of Stack
ELSEIF @stack[@stack.size-1] EQ 0
   STORE   tempSwap 99
   STORE   .
   RESTORE tempSwap 99
   MARK
# if Body,Body on top of Stack
ELSE
   STORE   tempSwap 98
   STORE   tempSwap 99
   RESTORE tempSwap 98
   RESTORE tempSwap 99
ENDIF
```

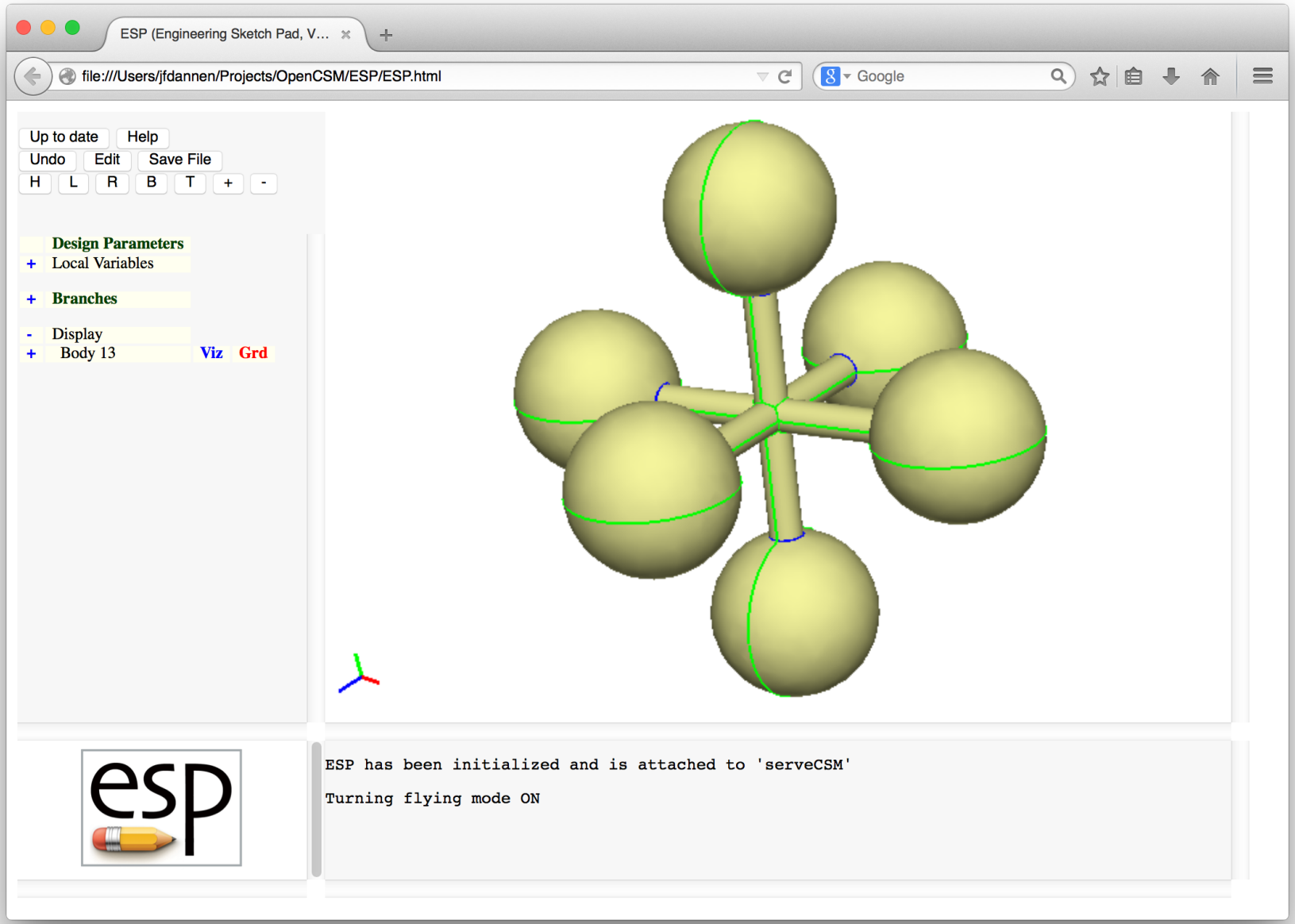
Example UDC — dumbbell.udc

```
# dumbbell

INTERFACE Lbar      in  0    # length of bar
INTERFACE Dbar      in  0    # diameter of bar
INTERFACE Dball     in  0    # diameter of balls
INTERFACE vol       out 0    # volume

ASSERT    ifpos(Lbar,1,0)   1
ASSERT    ifpos(Dbar,1,0)   1
ASSERT    ifpos(Dball,1,0)  1
SET       Lhalf    "Lbar / 2"

CYLINDER  -Lhalf  0  0  +Lhalf  0  0  Dbar
SPHERE    -Lhalf  0  0   Dball
UNION
SPHERE    +Lhalf  0  0   Dball
UNION

SET       vol    @volume

END
```

Example UDC — jack.csm

```
# jack

UDPARG  $/dumbbell  Lbar   5.0
UDPARG  $/dumbbell  Dball  1.0
UDPRIM  $/dumbbell  Dbar   0.2
SET     foo @@vol
STORE   dumbbell 0 1

RESTORE dumbbell
ROTATEY 90  0  0
UNION

RESTORE dumbbell
ROTATEZ 90  0  0
UNION

# show that vol was a local variable in .udc
ASSERT  ifnan(vol,1,0)  1
END
```

Example UDC — Jack



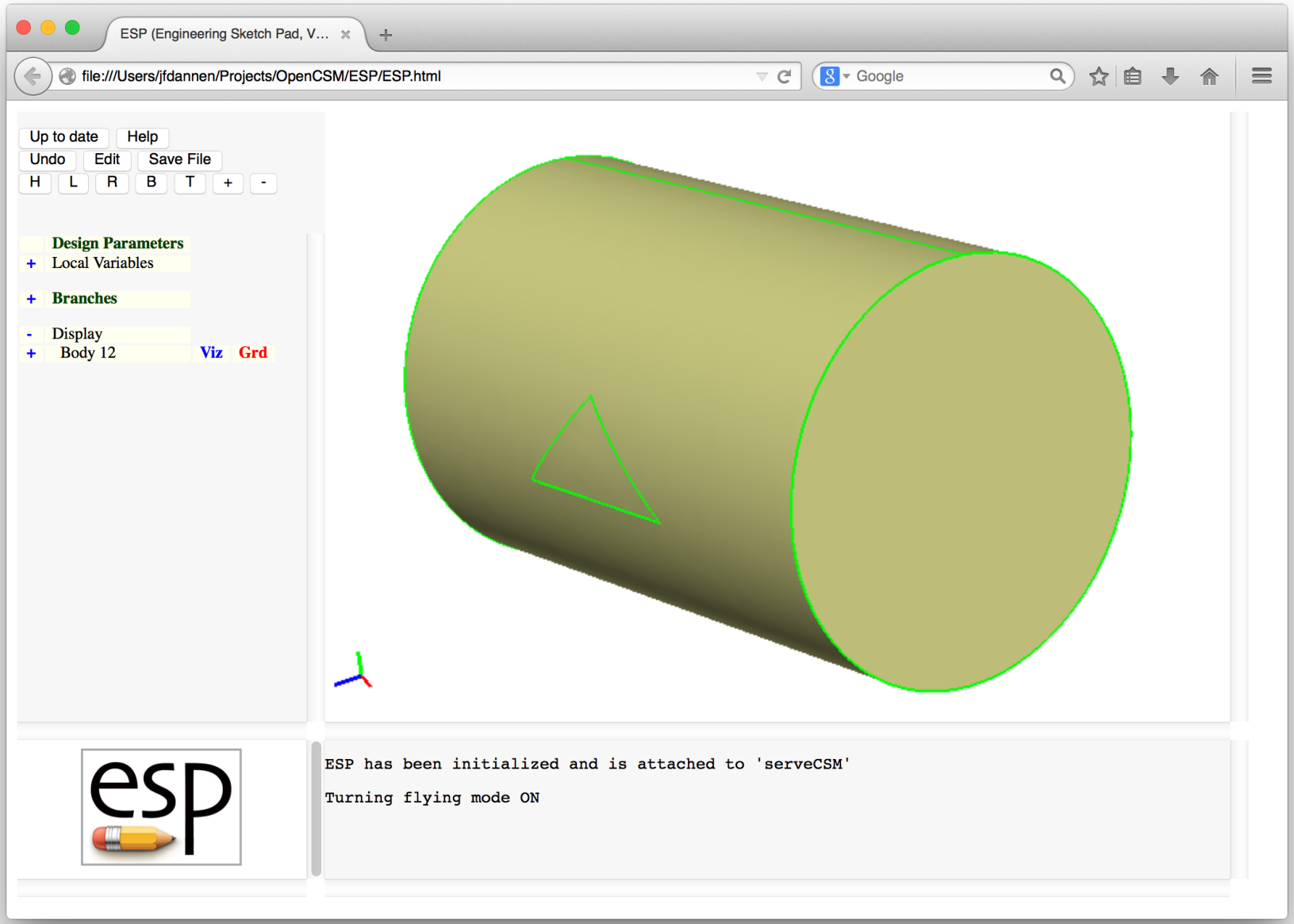
Example UDC — `cutter.udc`

```
# cutter

INTERFACE xx    in  0
INTERFACE yy    in  0
INTERFACE zbeg  in  0
INTERFACE zend  in  0

ASSERT    ifpos(xx.size-2,1,0)  1
ASSERT    ifzero(xx.size-yy.size,1,0)  1

SKBEG          xx[1]    yy[1]    zbeg
   PATBEG i xx.size-1
      LINSEG  xx[i+1]  yy[i+1]  zbeg
   PATEND
   LINSEG      xx[1]    yy[1]    zbeg
SKEND  1

EXTRUDE   0  0  zend-zbeg

END
```

Example UDC — scribeCyl.csm

```
# scribeCyl

DIMENSION xpoints   1  3
DIMENSION ypoints   1  3

SET       xpoints  "-1.;  1.;  .0;"
SET       ypoints  "-.5; -.5; +.5;"

CYLINDER  -3  0  0  +3  0  0  2
ROTATEX   90  0  0

UDPARG    $/cutter  xx    xpoints
UDPARG    $/cutter  yy    ypoints
UDPARG    $/cutter  zbeg  0
UDPRIM    $/cutter  zend  3
SUBTRACT

END
```

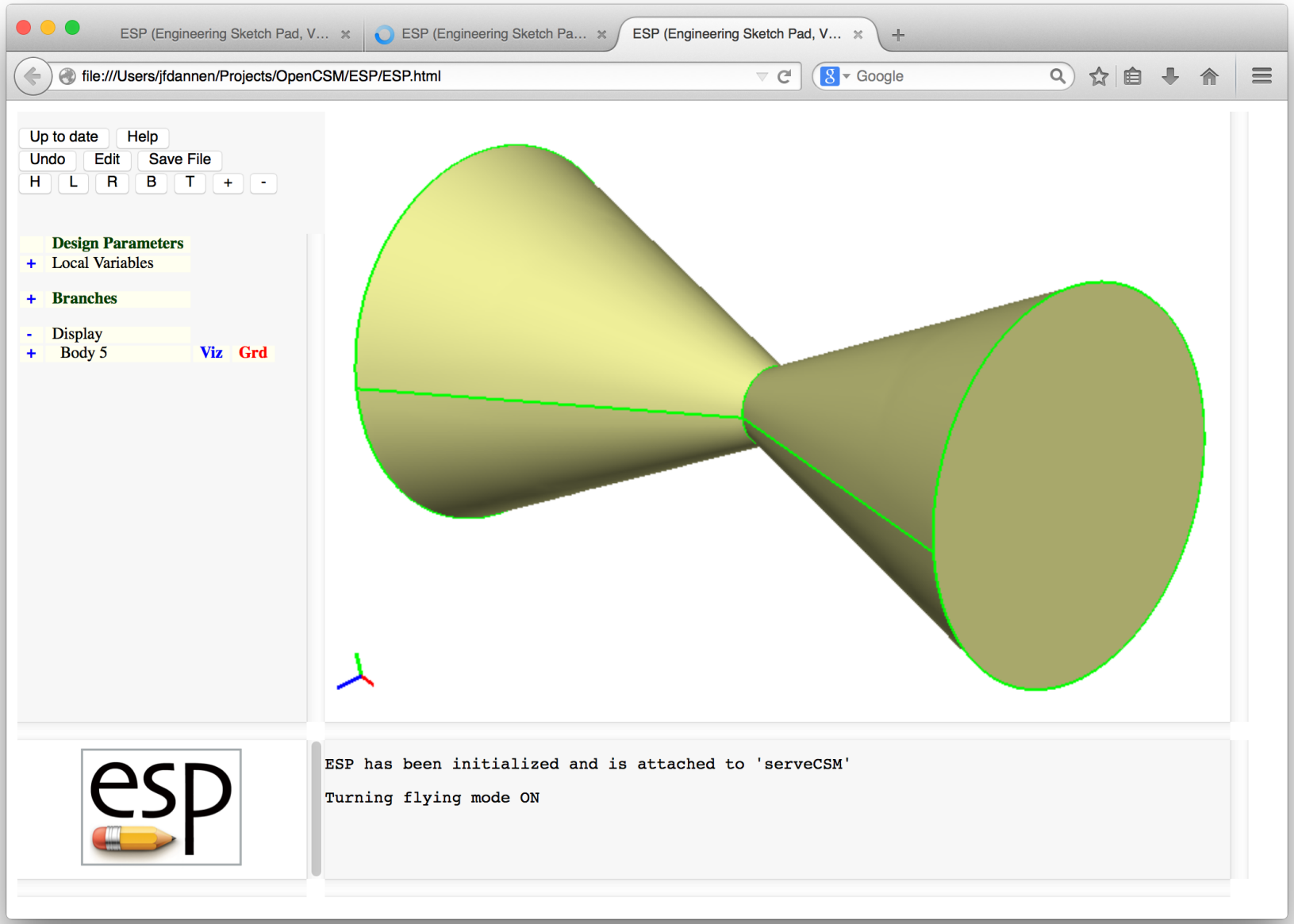
Example UDC — Scribed Cylinder



Homework Exercises

- Reflected cone
- Files in `$ESP_ROOT/training/ESP/exercises/session05` will get you started

Reflected Cone (1)



Reflected Cone (2)

- Write `mirrorDup.udc` to
  - store a copy of the Body on the top of the Stack
  - mirror the Body across a plane whose normal vector and distance from the origin are given
  - union the original and mirrored Bodys
- Apply `mirrorDup.udc` to a cone
  - cone base at $(5, 0, 0)$
  - cone vertex at $(0, 0, 0)$
  - cone diameter is 4
  - reflection across a plane at $x = 1$