Engineering Sketch Pad (ESP)

Training Session 3
Solids Fundamentals (2)

John F. Dannenhoffer, III
jfdannen@syr.edu
Syracuse University

Bob Haimes
haimes@mit.edu
Massachusetts Institute of Technology

updated for v1.18

Overview

- Miscellaneous Branches
- Grown Bodys
  - EXTRUDE
  - REVOLVE
  - RULE
  - BLEND
- Creating a Waffle
  - UDPRIM WAFFLE
- Homework Exercises

Miscellaneous Branches (1)

- `SET` — set the value of a local variable to the given expression
- `MARK` — push a Mark onto the stack
- `GROUP` — put all Bodys on stack since the Mark (or beginning) into a Group
  - transformations are applied to all Bodys in a Group
  - `STORE` operation stores all Bodys in Group
- `SELECT` — select entity for which @-parameters are evaluated
  - see "help" for details
- `PROJECT` — find the first projection from a given point (in space) in a given direction

Miscellaneous Branches (2)

- `STORE` — remember the identity of the Group (of Bodys) on the top of the stack
  - each storage location has a name and an optional index
  - depending on the value of `keep`, the Group/Body on the top of the stack is either kept (like a "copy") or popped off the stack (like a "cut")
  - Bodys can be popped off the Stack (and discarded) when the name is given as `.` (one Body), `..` (Bodys to Mark), or `...` (all Bodys)
  - this command is typically used in conjunction with the `RESTORE` primitive
- `DUMP` — write file that contains the Body (not Group) on the top of the stack
  - if `remove` is not zero, the Body is popped off the stack
  - if `toMark` is not zero, all Bodys since the Mark are written

Miscellaneous Branches (3)

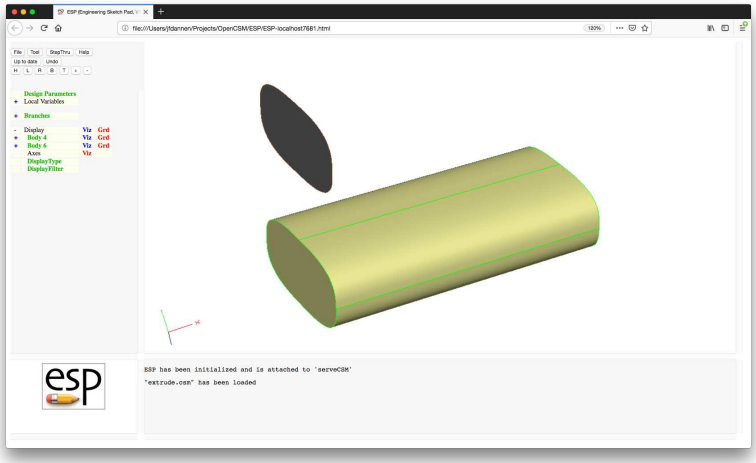
- The types of files that can be written by `DUMP` include:
  - `.brep` or `.BREP` — `OpenCASCADE` output
  - `.bstr` or `.BSTR` — binary stereolithography output
  - `.egads` or `.EGADS` —`EGADS` output
  - `.egg` or `.EGG` — EGG restart output
  - `.igs` or `.IGS` — IGES output
  - `.sens` or `.SENS` — sensitivity information
  - `.step` or `.STEP` — STEP output
  - `.stl` or `.STL` — ASCII stereolithography output
  - `.stp` or `.STP` — STEP output
  - `.tess` or `.TESS` — ASCII tessellation output
  - `.ugrid` or `.UGRID` — ASCII AFLR3 output

Grown Primitives (from SheetBodys)

- Pops one or more SheetBodys from the stack
- Pushes the resultant Body onto the stack
- Supported grown features include:
  - `EXTRUDE` — in a given direction for a given distance
  - `REVOLVE` — around a given axis for a given angular displacement
  - `RULE` — connect all the SheetBodys back to the Mark by straight lines
    - the first and/or last Sketch can be a NodeBody
  - `BLEND` — connect all the SheetBodys back to the Mark with smooth curves
    - the first and/or last Sketch can be a NodeBody
    - at the bounding Nodes, the user can specify the radius of curvature in two orthogonal directions
  - `SWEEP` — a SheetBody along a given Wire
    - this is often problematic in `OpenCASCADE`
  - `LOFT` — similar to `BLEND`, but with less control

Grown Primitive — EXTRUDE

Note: Original Sketch (SheetBody) and result of EXTRUDE are shown

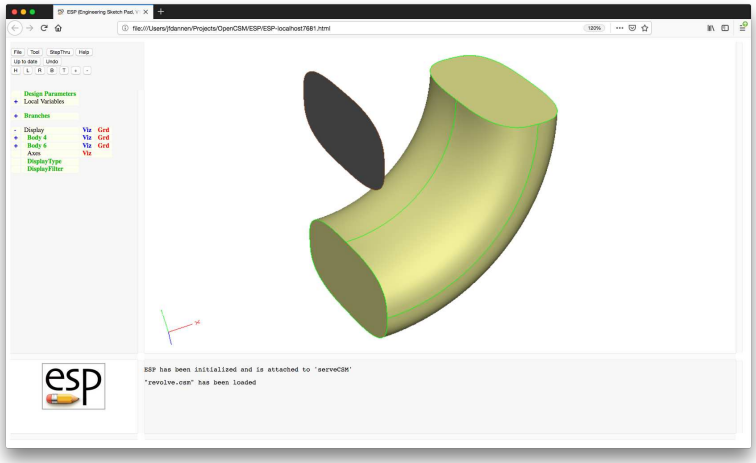


```
# extrude

UDPRIM  supell rx 2 ry_n 1 ry_s 1 n 3
ROTATEY 90 0 0
STORE  sections

RESTORE sections
TRANSLATE 0 4 0

RESTORE sections
EXTRUDE 8 0 0

END
```

- Face-order is: (1) orig Sketch, (2) copy of Sketch, (3) Face from first Sketch Edge, (4) Face from second Sketch Edge, . . .

Grown Primitive — REVOLVE

Note: Original Sketch (SheetBody) and result of REVOLVE are shown

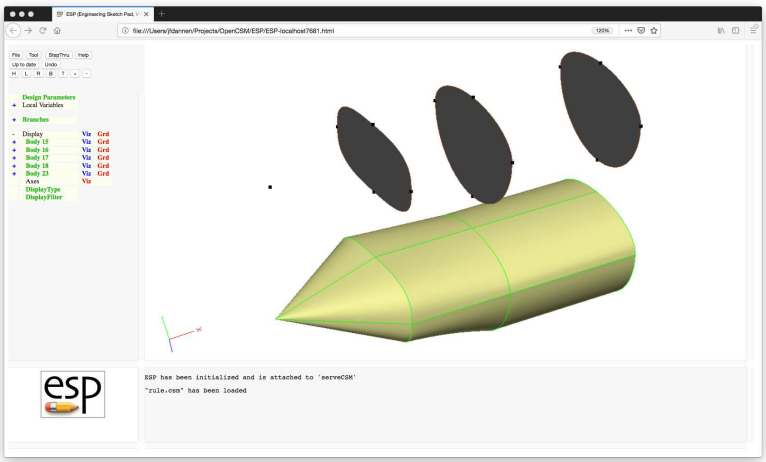


```
# revolve

UDPRIM  supell rx 2 ry_n 1 ry_s 1 n 3
ROTATEY 90 0 0
STORE  sections

RESTORE sections
TRANSLATE 0 4 0

RESTORE sections
REVOLVE 0 4 0  0 0 1  90

END
```

- Face-order is: (1) orig Sketch, (2) copy of Sketch, (3) Face from first Sketch Edge, (4) Face from second Sketch Edge, . . .

Grown Primitive — RULE

Note: Original Sketches (SheetBodys) and result of RULE are shown

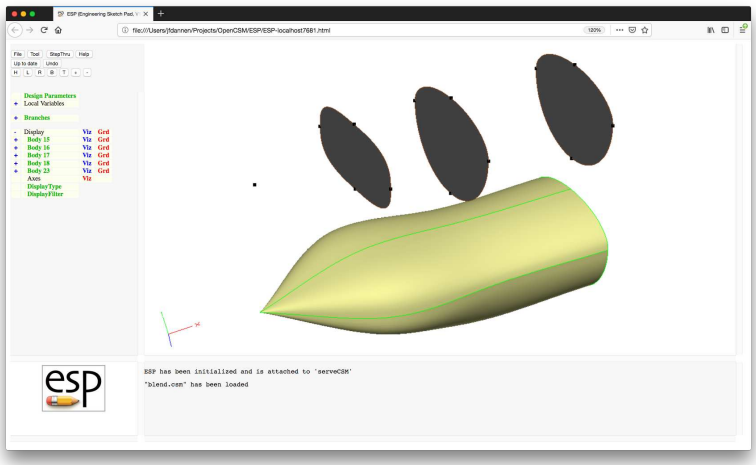



```
# rule

MARK
   POINT  0 0 0

   UDPRIM  supell rx 2 ry_n 1 ry_s 1 n 3
   ROTATEY 90 0 0
   TRANSLATE 3 0 0

   UDPRIM  supell rx 2 ry_n 1 ry_s 2
   ROTATEY 90 0 0
   TRANSLATE 6 0 0

   UDPRIM  supell rx 2 ry_n 1 ry_s 2
   ROTATEY 90 0 0
   TRANSLATE 10 0 0
GROUP
STORE  sections

RESTORE sections
TRANSLATE 0 4 0

MARK
   RESTORE sections
RULE

END
```

- Face-order on next slide

Grown Primitive — BLEND

Note: Original Sketches (SheetBodys) and result of BLEND are shown



```
# blend

MARK
   POINT  0 0 0

   UDPRIM  supell rx 2 ry_n 1 ry_s 1 n 3
   ROTATEY 90 0 0
   TRANSLATE 3 0 0

   UDPRIM  supell rx 2 ry_n 1 ry_s 2
   ROTATEY 90 0 0
   TRANSLATE 6 0 0

   UDPRIM  supell rx 2 ry_n 1 ry_s 2
   ROTATEY 90 0 0
   TRANSLATE 10 0 0
GROUP
STORE  sections

RESTORE sections
TRANSLATE 0 4 0

MARK
   RESTORE sections
BLEND

END
```

- Face-order on next slide

Face-order for RULE and BLEND

- (1) first Sketch (or empty if POINT)
- (2) last Sketch (or empty if POINT)
- (3) Face from first Sketch Edge between first and second Sketches
- (4) Face from first Sketch Edge between second and third Sketches
- ...
- (n) Face from second Sketch Edge between first and second Sketches
- ...

RULE and BLEND

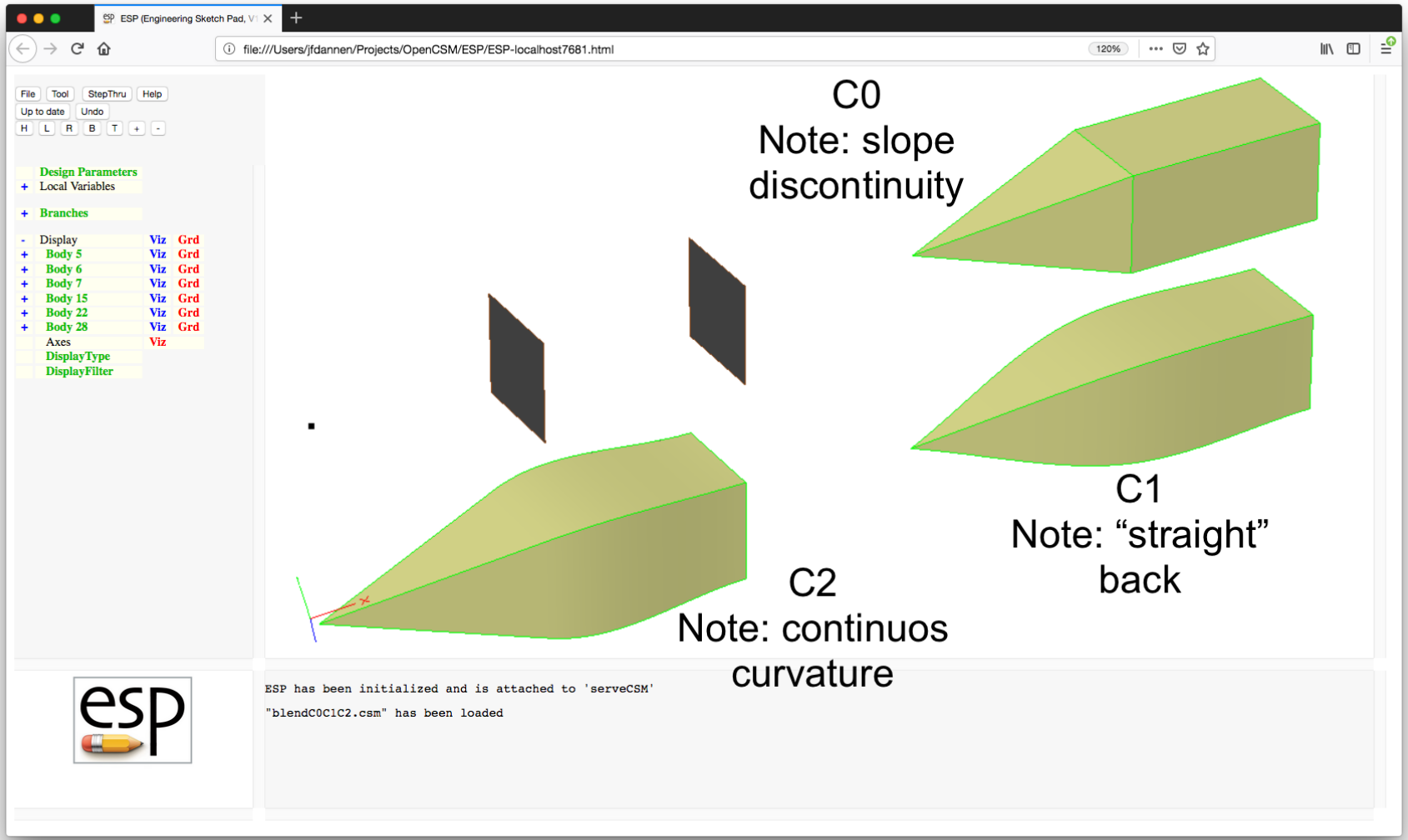
- RULE and BLEND require that all SheetBodys have the same number of Segments, ordered in the same way
  - new Faces are made by combining all the first Segments, ...
- BLEND allows user-selectable continuity in blend direction
  - C2 - curvature continuity (the default)
  - C1 - slope continuity (obtained with Face repeated once)
  - C0 - value continuity (obtained with Face repeated twice)
- SheetBodys can be automatically reordered to help eliminate twist by setting **reorder** to a non-zero value
  - positive to start from first Sketch
  - negative to start from last Sketch
- Users can manually reorder SheetBodys with the **reorder** command (applied to a SheetBody)
  - Reordering only changes the order of Segments, not their shapes

BLEND Continuity (1)

```
# blendC0C1C2

# original sketches (top left)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
GROUP
TRANSLATE -3 +1 0

# Body with C0 at second sketch (top rite)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1
   UDPRIM box  dy 1  dz 1
   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
BLEND
TRANSLATE +3 +1 0
```

```
# Body with C1 at second Sketch (bottom left)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1
   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
BLEND
TRANSLATE -3 -1 0

# Body with C2 at second Sketch (bottom rite)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
BLEND
TRANSLATE +3 -1 0

END
```

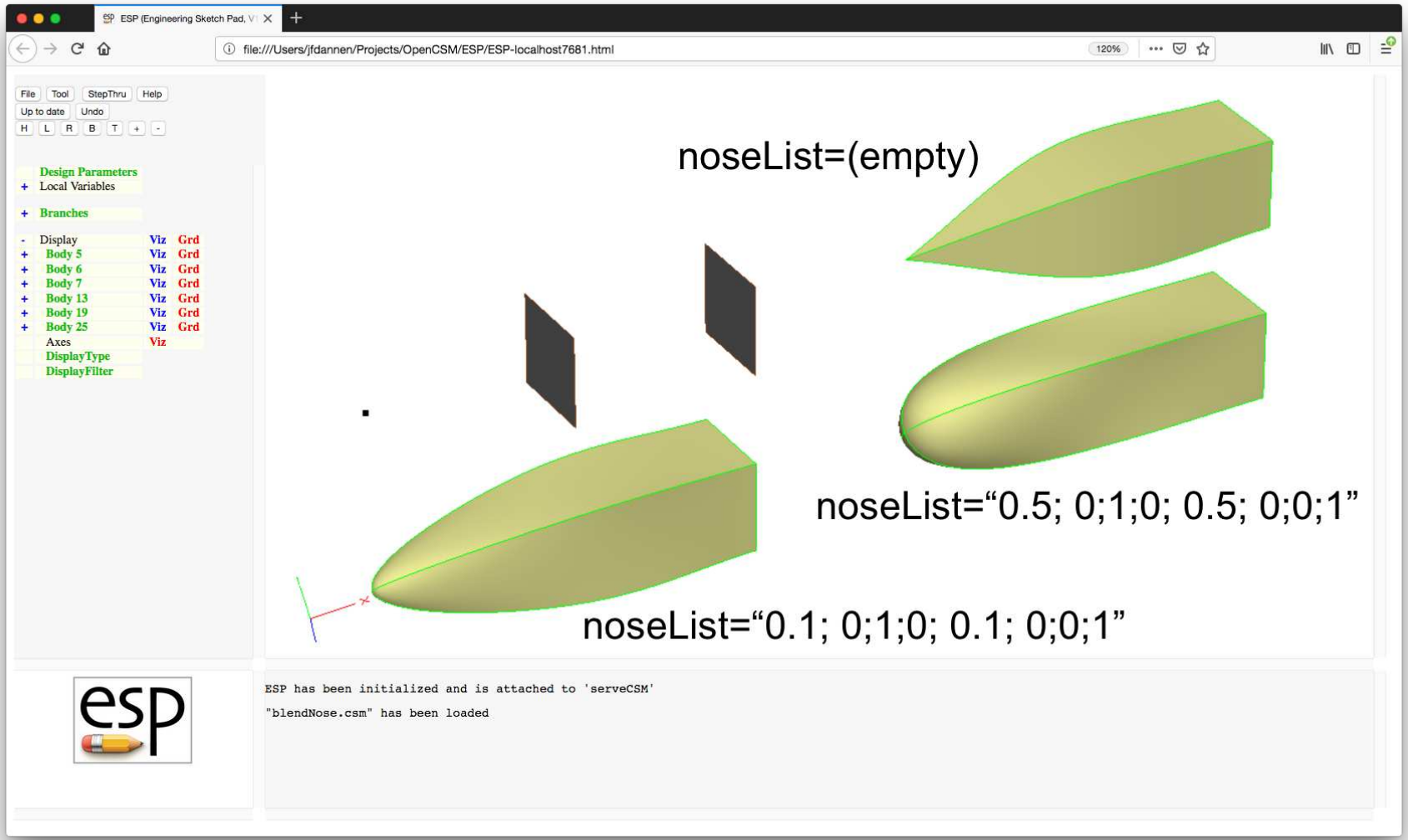
BLEND Continuity (2)



BLEND Nose/Tail Treatment (1)

```
# blendC0C1C2

# original sketches (top left)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
GROUP
TRANSLATE -3 +1 0

# Body with pointed nose (top rite)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
BLEND
TRANSLATE +3 +1 0

# Body with slightly rounded nose (bottom left)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
BLEND "0.1; 0;1;0; 0.1; 0;0;1"
TRANSLATE -3 -1 0

# Body with rounded nose (bottom rite)
MARK
   POINT  -2  0  0

   UDPRIM box  dy 1  dz 1

   UDPRIM box  dy 1  dz 1
   TRANSLATE +2 0 0
BLEND "0.5; 0;1;0; 0.5; 0;0;1"
TRANSLATE +3 -1 0

END
```

BLEND Nose/Tail Treatment (2)



Building a Waffle (1)

- Called with `.csm` statement:
  `UDPRIM waffle depth <number> filename <name_of_file>`
- Valid statements in file are:
  - `CPOINT` — create a construction point (not in final waffle)
  - `CLINE` — create a construction line (not in final waffle)
  - `POINT` — create a waffle point
  - `LINE` — create one or more waffle segments
  - `PATBEG/PATEND` — create a pattern (loop)
- Keywords can be in lowercase or UPPERCASE
- Coordinates of existing point `<pname>` are given by
  - `x@<pname>` and `y@<pname>`

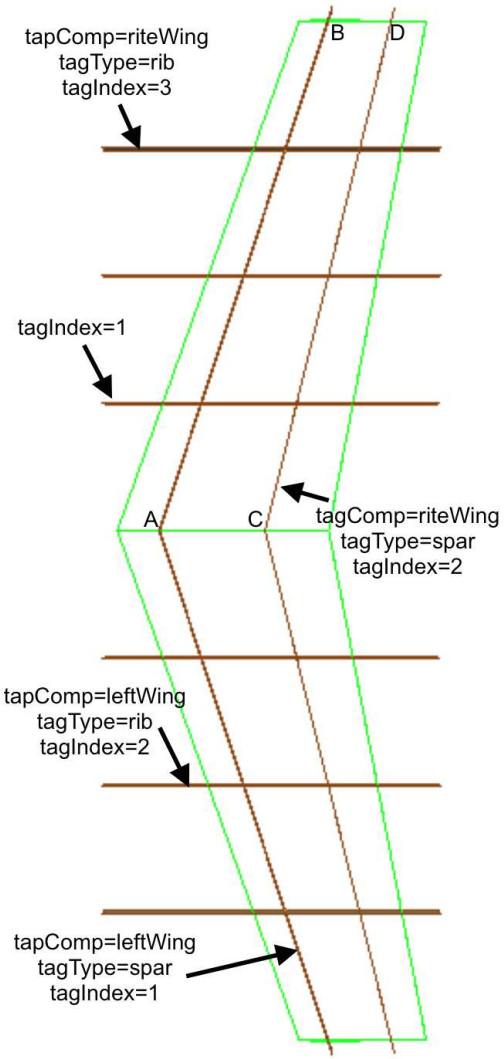
Building a Waffle (2)

- Variants of `CPOINT` and `POINT`
  - `POINT <pname> AT <xloc> <yloc>`
    - create point at `<xloc,yloc>`
  - `POINT <pname> ON <lname> FRAC <fracDist>`
    - creates point on `<lname>` at given fractional distance
  - `POINT <pname> ON <lname> XLOC <x>`
    - creates point on `<lname>` at given `<x>`
  - `POINT <pname> ON <lname> YLOC <y>`
    - creates point on `<lname>` at given `<y>`
  - `POINT <pname> ON <lname> PERP <pname2>`
    - creates point on `<lname>` that is closest to `<pname2>`
  - `POINT <pname> ON <lname> XSECT <lname2>`
    - creates point at intersection of `<lname>` and `<lname2>`
  - `POINT <pname> OFF <lname> <dist> <pname2>`
    - creates point `<dist>` to the left of `<lname>` at `<pname2>`

Building a Waffle (3)

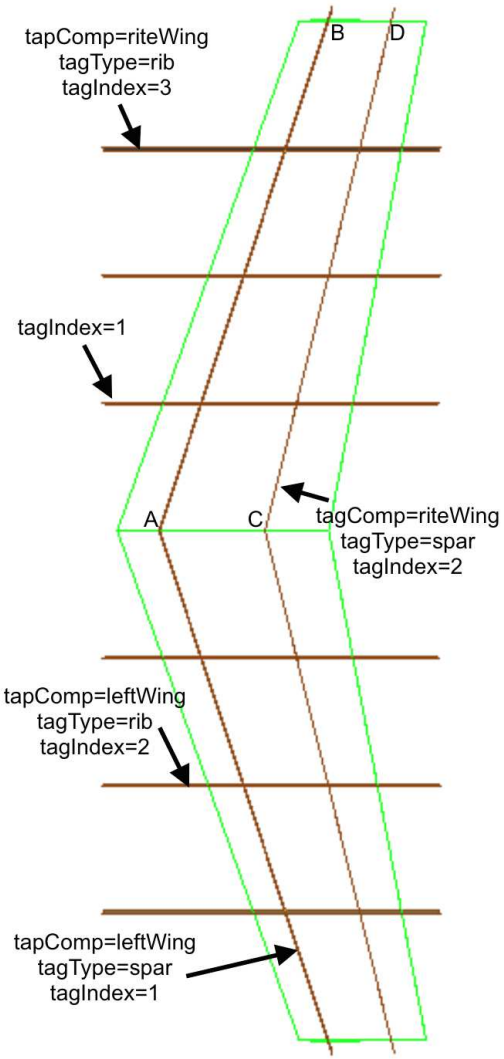
- Variants of `CLINE` and `LINE`
  - `LINE . <pname1> <pname2> <attrName1=attrValue1>...`
    - creates unnamed line between `<pname1>` and `<pname2>` with given attribute(s) (if any)
  - `LINE <lname> <pname1> <pname2> <attrName1=attrValue1>`
    - creates line named `<lname>` between `<pname1>` and `<pname2>` with given attribute(s) (if any)

Waffle for wing3 (1)



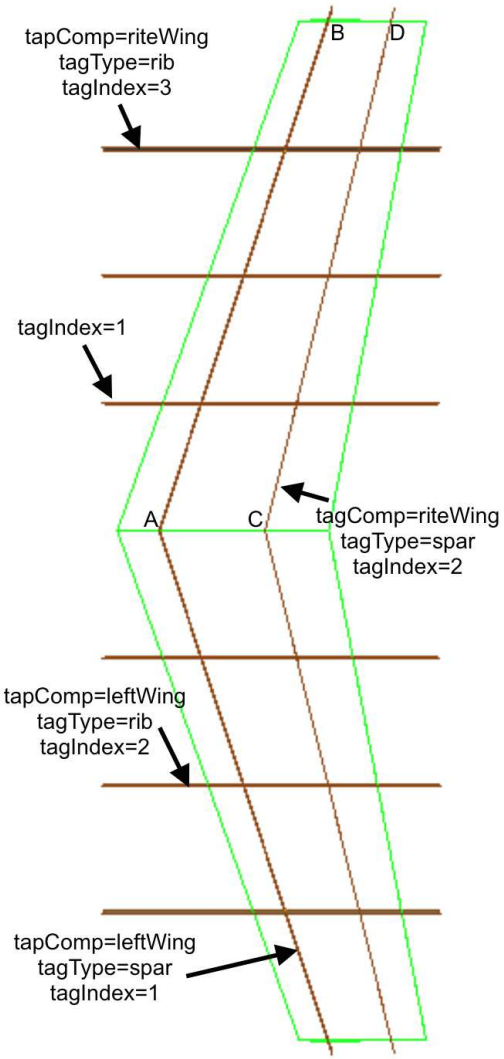

```
SET       xmin        @xmin-0.1
SET       xmax        @xmax+0.1
SET       ymin        0
SET       ymax        @ymax+0.1
SET       zmin        @zmin-0.1
SET       zmax        @zmax+0.1
STORE     .

UDPARG    waffle      depth wing:nrib     # ensures rebuild
UDPARG    waffle      depth wing:spar1
UDPARG    waffle      depth wing:spar2
UDPRIM    waffle      depth zmax-zmin  filename <<

# construction lines for spars
CPOINT A    AT          0+wing:spar1*croot 0
CPOINT B    AT  wing_xtip+wing:spar1*ctip  wing_ytip
CPOINT C    AT          0+wing:spar2*croot 0
CPOINT D    AT  wing_xtip+wing:spar2*ctip  wing_ytip

CLINE  AB       A  B
CLINE  CD       C  D
```

Waffle for `wing3` (2)




```
# rite spars
POINT  E   ON  AB   YLOC  ymin
POINT  F   ON  AB   YLOC  ymax
LINE   EF  E   F  tagComp=riteWing  tagType=spar tagIndex=1

POINT  G   ON  CD   YLOC  ymin
POINT  H   ON  CD   YLOC  ymax
LINE   GH  G   H  tagComp=riteWing  tagType=spar tagIndex=2

# rite ribs
PATBEG  irib  wing:nrib
    CPOINT  I  AT  xmin  wing_ytip*irib/(wing:nrib+1)
    CPOINT  J  AT  xmax  y@I
    LINE    .  I J tagComp=riteWing tagType=rib ...
                          tagIndex=!val2str(irib,0)
PATEND
```

Waffle for `wing3` (3)

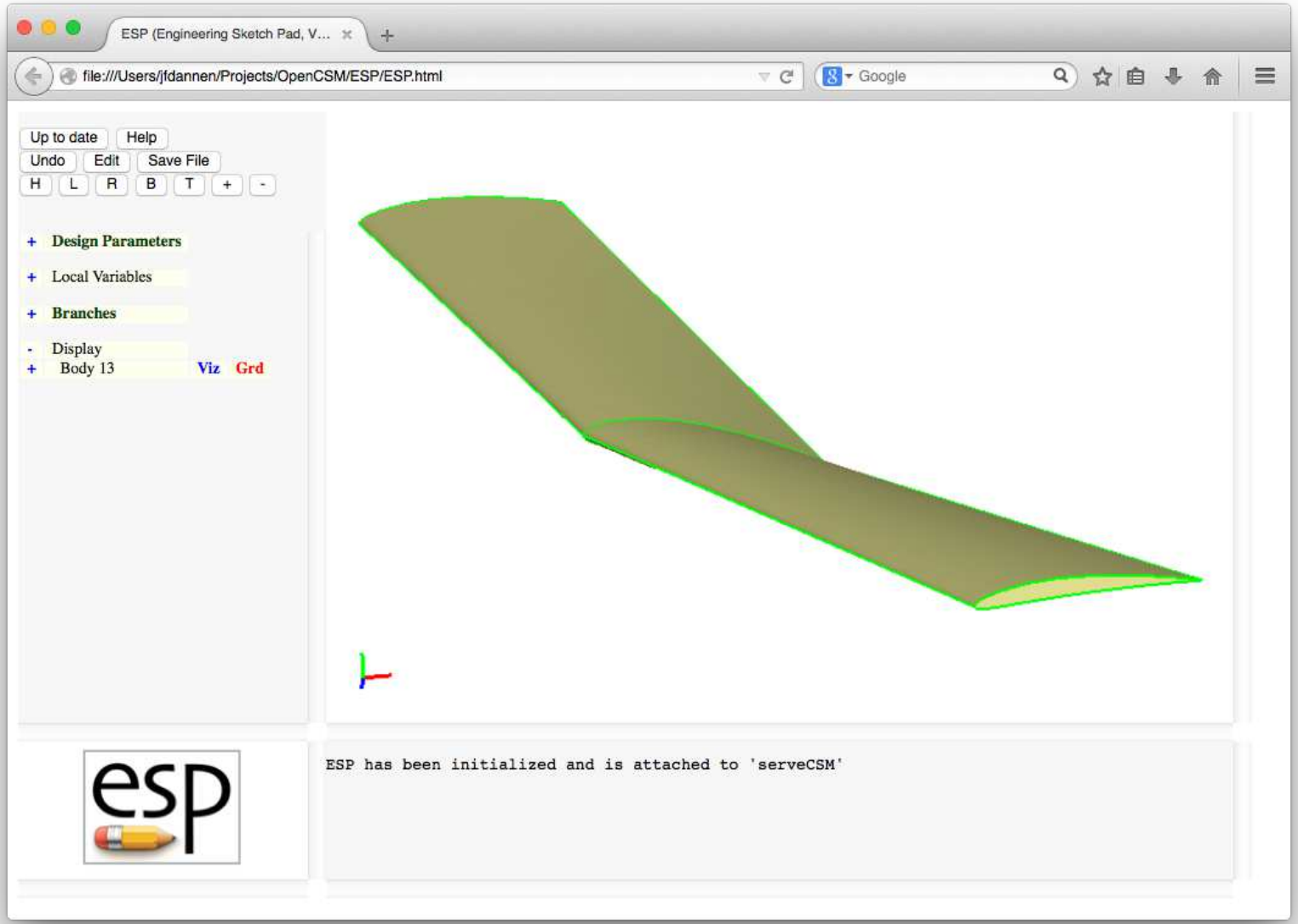



```
# left spars
POINT  E   AT  x@E  -y@E
POINT  F   AT  x@F  -y@F
LINE   EF  E   F  tagComp=leftWing  tagType=spar tagIndex=1

POINT  G   AT  x@G  -y@G
POINT  H   AT  x@H  -y@H
LINE   GH  G   H  tagComp=leftWing  tagType=spar tagIndex=2

# left ribs
PATBEG  irib  wing:nrib
    CPOINT  I  AT  xmin  -wing_ytip*irib/(wing:nrib+1)
    CPOINT  J  AT  xmax  y@I
    LINE    .  I J tagComp=leftWing tagType=rib ...
                          tagIndex=!val2str(irib,0)

PATEND
>>
```

Homework Exercises

- Simple wing
- Simple fuselage
  - OML (outer mold line)
  - structure
- Starter files are in `$ESP_ROOT/training/ESP/data/session03`

Simple Wing (1)



Simple Wing (2)

Xroot	X-coordinate of root leading edge	0.00
Yroot	Y-coordinate of root leading edge	0.00
Zroot	Z-coordinate of root leading edge	0.00
croot	chord of root	2.00
troot	thickness/chord of root	0.12
mroot	camber/chord of root	0.04
aroot	angle of attack of root (deg)	7.50
Xtip	X-coordinate of tip leading edge	0.50
Ytip	Y-coordinate of tip leading edge	0.25
Ztip	Z-coordinate of tip leading edge	8.00
ctip	chord of tip	1.75
ttip	thickness/chord of tip	0.08
mtip	camber/chord of tip	0.04
atip	angle of attack of tip (deg)	-5.00

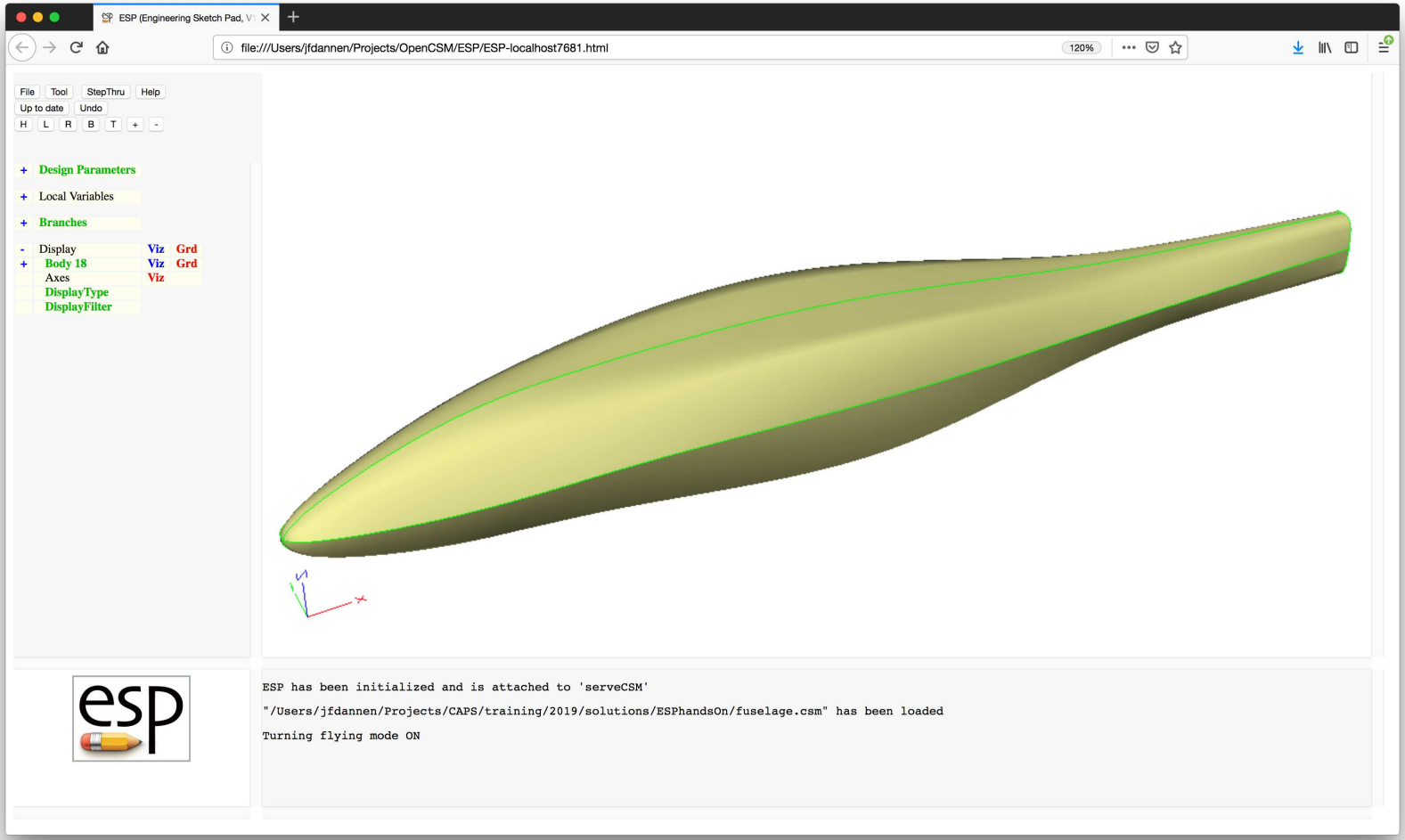
Simple Wing (3)

- What happens if you switch from `RULE` to `BLEND`?
- What happens if we change the sequence of transformations from `SCALE`, `ROTATEZ`, `TRANSLATE` to `ROTATEZ`, `SCALE`, `TRANSLATE`?
- What happens if we do the `TRANSLATE` first?
- Could you change the Design Parameters to `area`, `aspectRatio`, `taperRatio`, `sweep`, and `twist`?

$$AR = \frac{b^2}{S} \qquad S = b(c_{\text{tip}} + c_{\text{root}})/2 \qquad \tau = \frac{c_{\text{tip}}}{c_{\text{root}}}$$

Simple Fuselage (1)

- Fuselage by blending a series of super-ellipses (SUPELLs), where the dimensions of the cross-sections are provided in arrays



Simple Fuselage (2)

xloc	width	zcent	height	power
0.0	0.0	0.0	0.0	2
1.0	1.0	0.1	1.0	2
4.0	1.6	0.4	2.0	3
8.0	1.6	0.4	2.0	3
12.0	1.0	0.3	1.2	2
16.0	0.8	0.2	0.4	2

Simple Fuselage (3)

- Can you make the radius at the nose 0.2 in a top view and 0.1 in a side view?
- Can you make the fuselage between the two sections whose power is 3 have a constant cross-section?
- Can you create a SheetBody that has a plane of symmetry and cross-sections at every y, starting at $y = 1/2$ and spaced with $\Delta y = 1$?
- Can you color the odd-numbered bulkheads red and even-numbered bulkheads blue?
- Can you color the Edges at the intersections of the symmetry plane and bulkheads white?

Simple Fuselage (4)