Engineering Sketch Pad (ESP)

Training Session 1.5
Solids Fundamentals (1)

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

Bob Haimes
haimes@mit.edu

Marshall Galbraith
galbramc@mit.edu

Massachusetts Institute of Technology

Overview

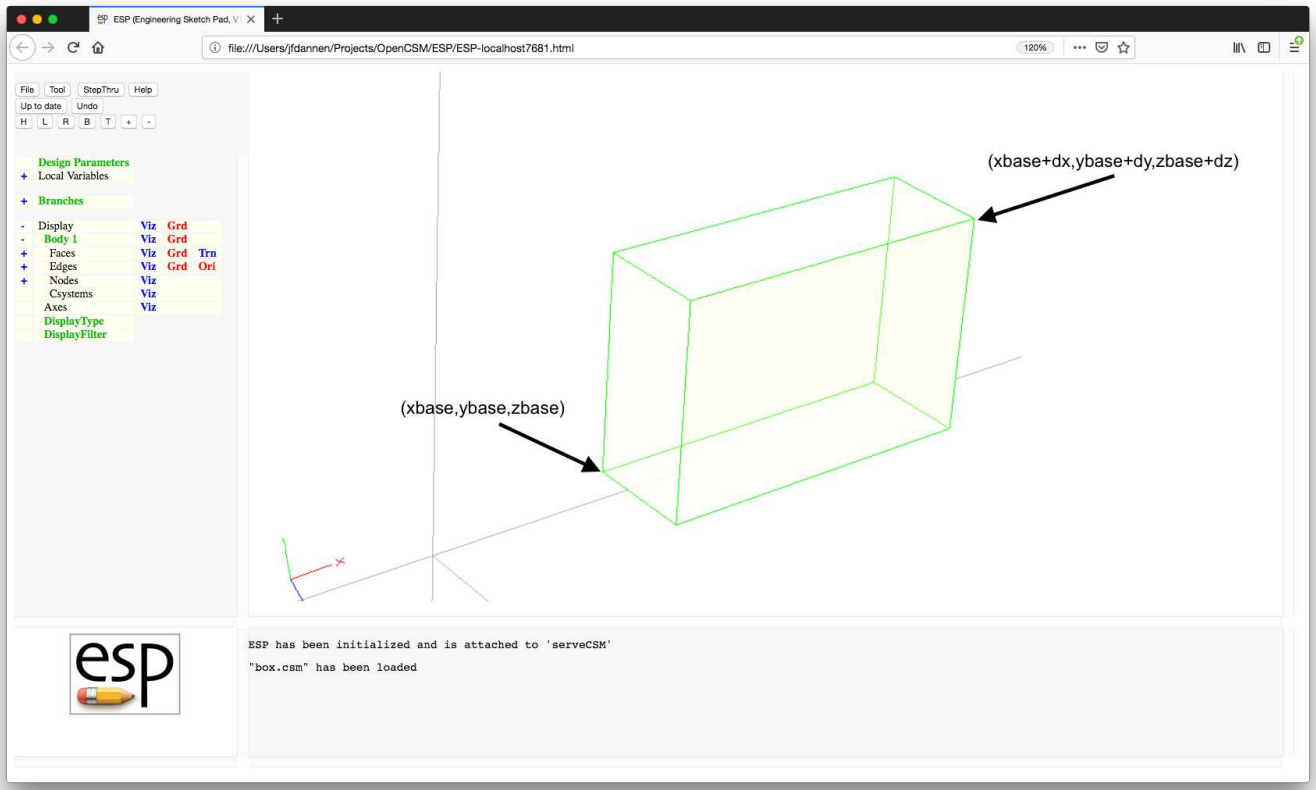
- Primitive Bodys
- Types of Bodys
- Transformations
- Boolean Operations
- User-defined Primitives and Functions
  - Difference Between UDPs and UDFs
  - Using UDPARG and UDPRIM Statements
- Creating Simple Cross-sections
- Creating a simple NodeBody, WireBody, SheetBody, and SolidBody

Standard Primitives (1)

- A primitive is a Branch that creates a new Body based solely upon its arguments
- Primitives pop NO entities from the stack
- Primitives push one Body onto the stack
- Built-in primitives include:
  - `POINT` — Node at given location
  - `BOX` — starting location and size
  - `SPHERE` — center and radius
  - `CYLINDER` — beginning center, ending center, and radius
  - `CONE` — base center, vertex, and radius
  - `TORUS` — center, orientation, major- and minor-radii

Standard Primitive — BOX

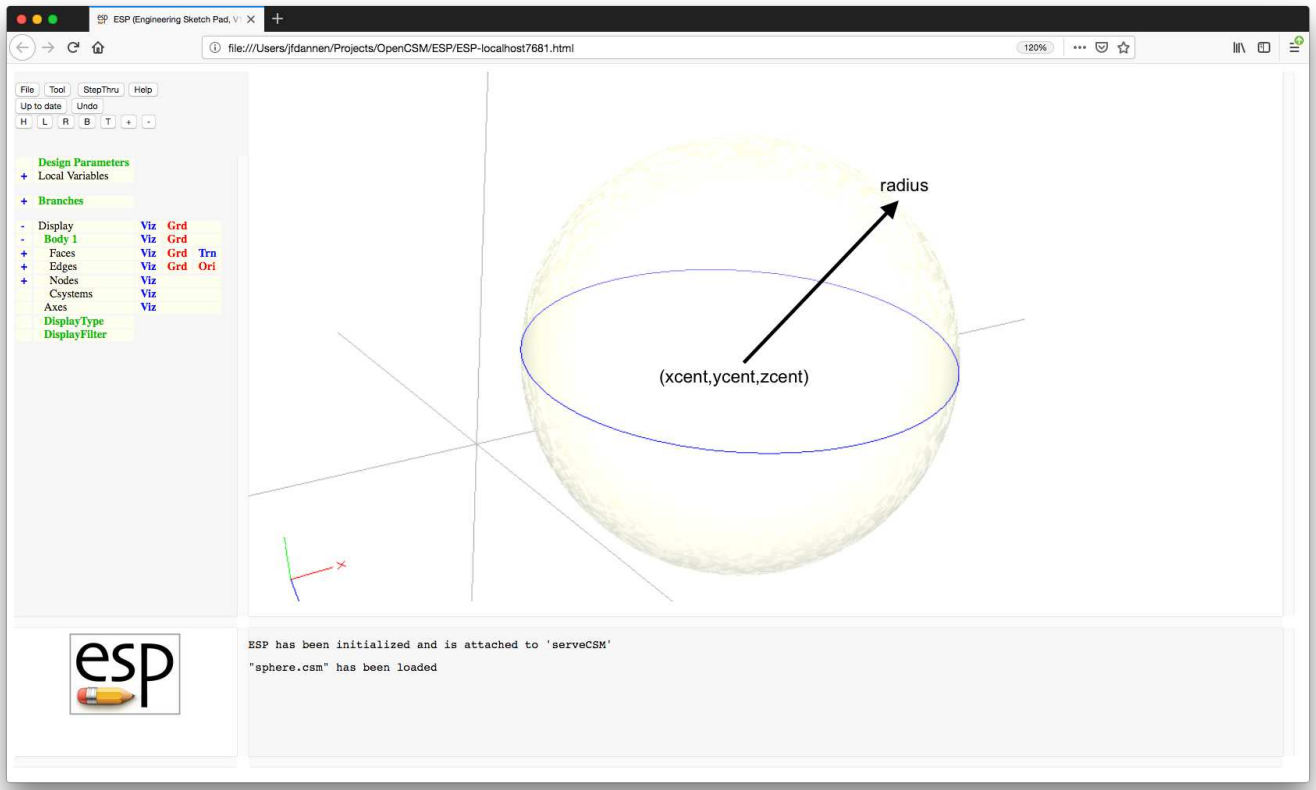
```
BOX xbase ybase zbase dx dy dz
```



- Face-order:
  - 1: x min
  - 2: x max
  - 3: y min
  - 4: y max
  - 5: z min
  - 6: z max

Standard Primitive — SPHERE

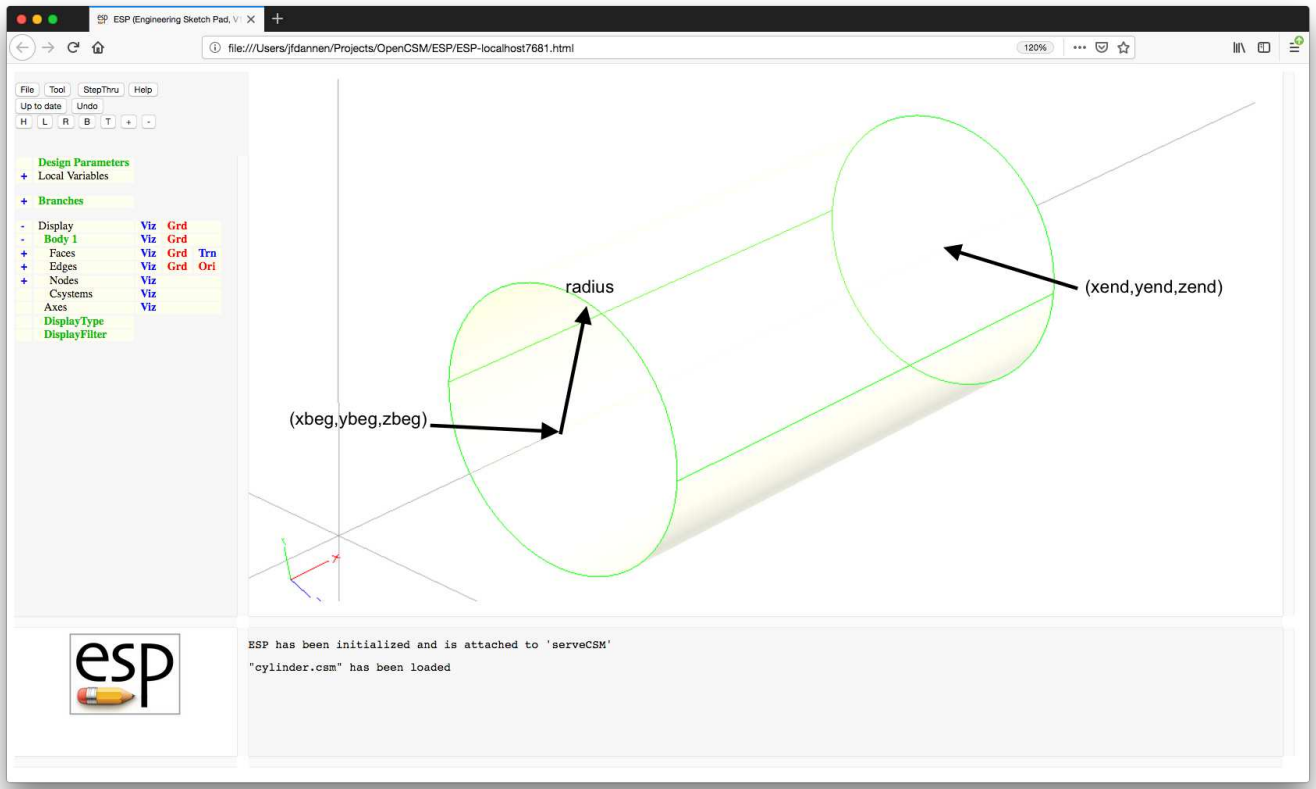
```
SPHERE xcent ycent zcent radius
```



- Face-order:
  - 1: y min
  - 2: y max

Standard Primitive — CYLINDER

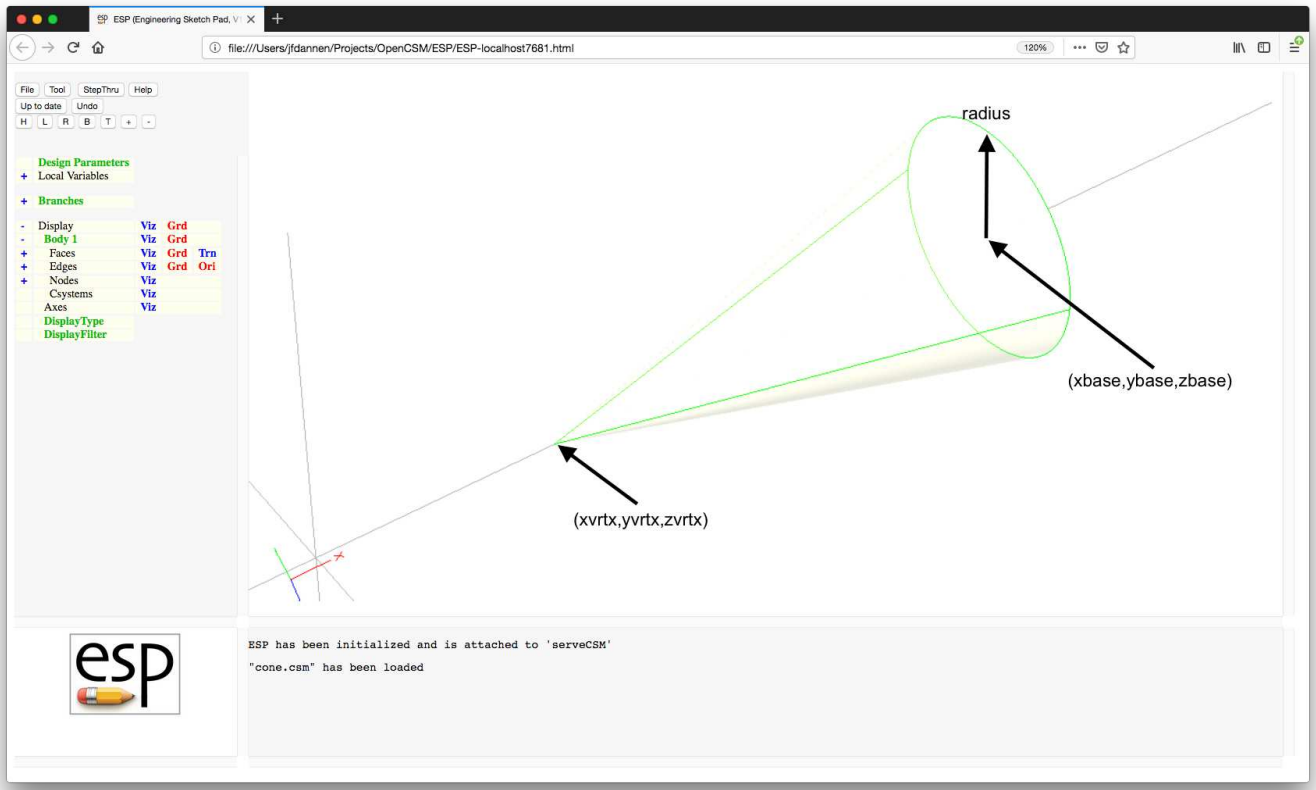
```
CYLINDER xbeg ybeg zbeg xend yend zend radius
```



- Face-order:
  - 1: beg
  - 2: end
  - 3: y min
  - 4: y max

Standard Primitive — CONE

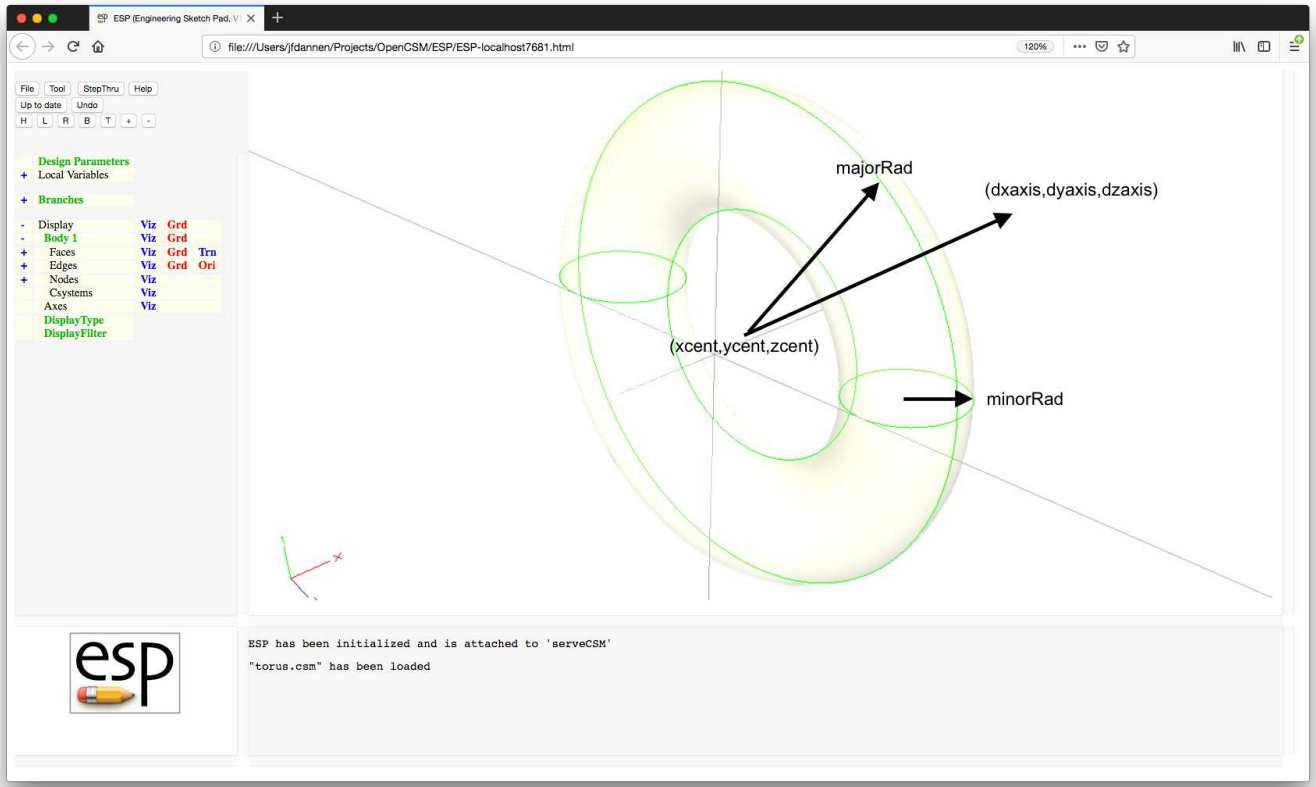
```
CONE xvrtx yvrtx zvrtx xbase ybase zbase radius
```



- Face-order:
  - 1: (empty)
  - 2: base
  - 3: y min
  - 4: y max

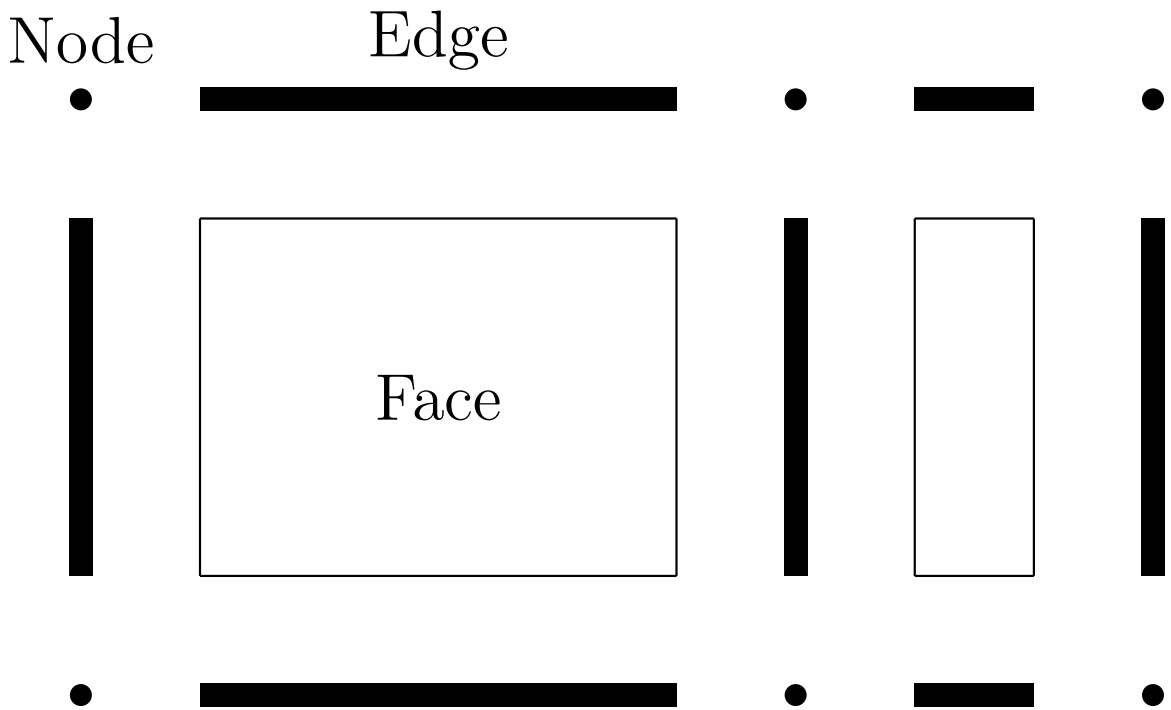
Standard Primitive — TORUS

`TORUS xcent ycent zcent dxaxis dyaxis dzaxis majorRad minorRad`



- Face-order:
  - 1: x min, y min
  - 2: x max, y max
  - 3: x max, y min
  - 4: x max, y max

BRep Nomenclature (1)



BRep Nomenclature (2)

- Node
  - a location in 3D space that serves as the terminus for one or more Edges
- Edge
  - is associated with a 3D curve (if not degenerate)
  - has a range of parametric coordinates, t, from t_{min} to t_{max}
  - the positive orientation goes from t_{min} to t_{max}
  - has a Node at t_{min} and t_{max}
  - if the Nodes at t_{min} and t_{max} are the same, the Edge forms a closed Loop (that is, is periodic) or is degenerate (if t_{min} equals t_{max}); otherwise it is open

BRep Nomenclature (3)

- Loop
  - free standing collection of one or more connected Edges with associated senses
  - if the Loop is closed, each of the corresponding Nodes is associated with exactly two Edges
  - if the Loop is open, the intermediate Nodes are each associated with two Edges and the Nodes at the ends each correspond to one Edge
  - the sense of the Loop is associated with the order of the Edges in the Loop and their associated senses
- Face
  - a surface bounded by one or more Loops with associated senses
  - there may be only one outer Loop (sense = 1) and any number of inner Loops (sense = -1)
  - associated Loops must be closed

BRep Nomenclature (4)

- Shell
  - a collection of one of more connected Faces
  - if all the Edges associated with a Shell are used by exactly two Faces in the Shell, the Shell is closed (manifold) and it segregates regions of 3D space; otherwise the Shell is open
- Body
  - a free-standing object

Types of Bodys

- SolidBody
  - Body that has an inside and outside
  - Bounded by a Shell
  - May contain one or more holes, each of which is defined by a Shell
- SheetBody
  - a single Shell that can be either non-manifold (open) or manifold (closed)
- WireBody
  - a single Loop
- NodeBody
  - a single Node (represented internally as a degenrate WireBody)

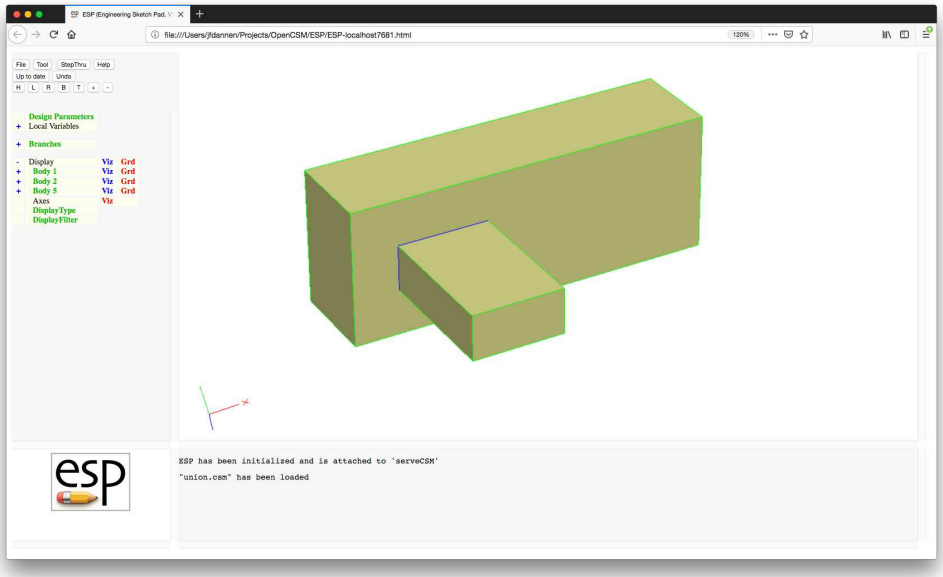
Boolean Operations (1)

- Pops two* Bodys off the stack
- Pushes the result onto the stack
- Supported Booleans include:
  - `UNION` — combine the input Bodys
  - `INTERSECT` — find the "common" parts of the input Bodys
    - if more than one Body results, order them based upon the `$order` argument and keep the `index`'th one
  - `SUBTRACT` — remove one Body from the other
    - if more than one Body results, order them based upon the `$order` argument and keep the `index`'th one

Boolean Operations (2)

- All Booleans have an optional tolerance that can be used in cases when `OpenCASCADE`'s geometric and topological tolerances cause an error
  - if the prescribed tolerance is positive, `OpenCSM` will first try its default tolerance; if unsuccessful, the tolerance will be loosened until success or until it reaches the specified tolerance
  - if the prescribed tolerance is nagative, `OpenCSM` will only try to use the absolute value of the given tolerance

Boolean Operation — UNION




```
# union

BOX      0 0 0  8 3 2
BOX      1 1 0  2 1 5

BOX      0 0 0  8 3 2
BOX      1 1 0  2 1 5

UNION

END
```

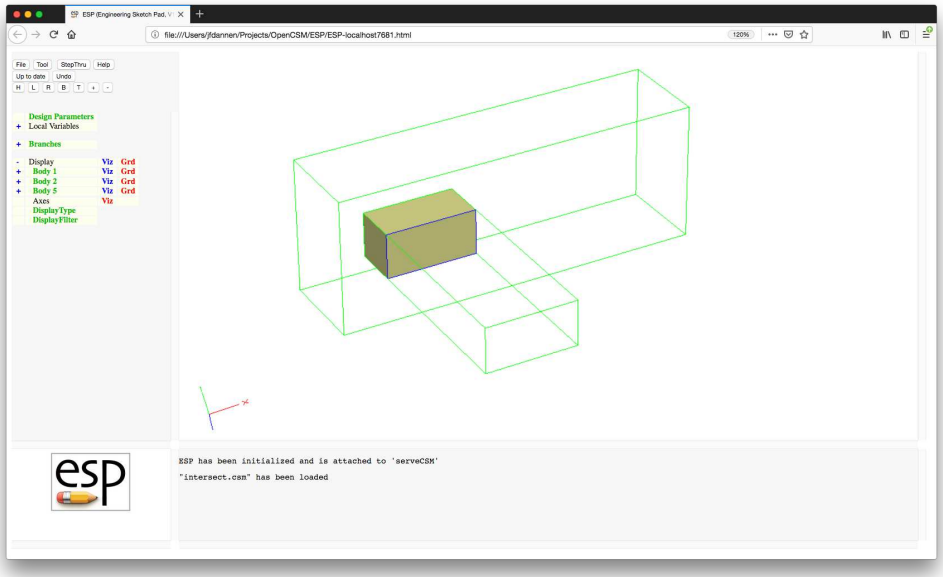
Special Rules for `UNION` (1)

- If `tomark` is 0 (not set)
  - If the two Bodys on the top of the Stack are both SolidBodys, a SolidBody that is the combination of its input Bodys is created
  - If the two Bodys on the top of the Stack are both SheetBodys, a SheetBody that is the combination of its input Bodys is created
  - Note: WireBodys cannot be UNIONed (use JOIN instead)
  - Note: the two Bodys on the top of the stack must be the same type

Special Rules for `UNION` (2)

- If `tomark` is not 0 (is set)
  - All the SolidBodys back to the Mark are combined
  - If other Body types are encountered, an error results
- If `trimList` is a list of six semicolon-separated numbers and `tomark` is not set and the Bodys on the top of the Stack are both SolidBodys, then the `UNION` is trimmed to only keep the part of Body2 that contains the `trimList`

Boolean Operation — INTERSECT

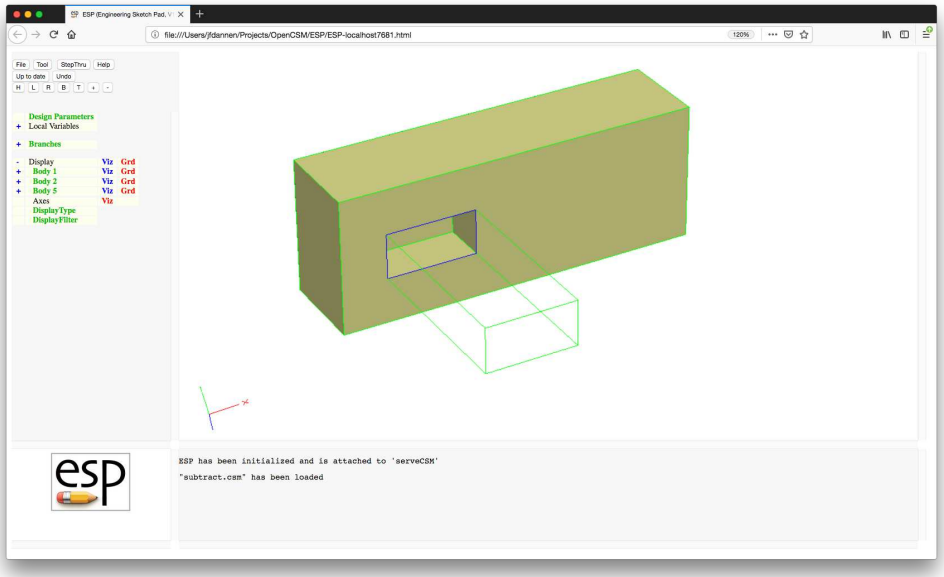



```
# intersect

BOX      0 0 0  8 3 2
BOX      1 1 0  2 1 5

BOX      0 0 0  8 3 2
BOX      1 1 0  2 1 5

INTERSECT

END
```

Special Rules for INTERSECT

- If both Bodys on the top of the Stack are SolidBodys
  - a SolidBody that is the common part of its inputs is created
- If one Body on the top of the Stack is a SolidBody and the other is a SheetBody
  - a SheetBody is created that is the part of the input SheetBody that is inside the SolidBody
- If one Body on the top of the Stack is a SolidBody and the other is a WireBody
  - a WireBody is created that is the part of the input WireBody that is inside the SolidBody
- Other combinations of input Bodys are not allowed

Boolean Operation — SUBTRACT




```
# subtract

BOX      0 0 0  8 3 2
BOX      1 1 0  2 1 5

BOX      0 0 0  8 3 2
BOX      1 1 0  2 1 5

SUBTRACT

END
```

Special Rules for `SUBTRACT` (1)

- Call the last Body on the Stack Body2 and the next-to-last Body Body1
- If Body1 and Body2 are both SolidBodys
  - create a SolidBody that is the part of Body1 that is outside Body2
- If Body1 is a SheetBody and Body2 is a Solid Body
  - create a SheetBody that is the part of Body1 that is outside Body2

Special Rules for SUBTRACT (2)

- If Body1 is a SolidBody and Body2 is a SheetBody
  - create a SolidBody that is the same shape as Body1, but which is scored with the intersection of Body2 (ie, new Nodes and Edges are created), only if the scoring completely cuts Body1
- If Body1 is a SheetBody and Body2 is a SheetBody
  - create a SheetBody that is the same shape as Body1, but which is scored with the intersection of Body2 (ie, new Nodes and Edges are created), only if the scoring completely cuts Body1
- Other combinations of input Bodys are not allowed

Other Boolean-like Operations (1)

- `JOIN` — two SolidBodys at common Faces, two SheetBodys at common Edges, or two WireBodys at common Node
  - this is much more efficient (and robust) than UNION
- `CONNECT` — two Bodys by creating bridging Faces
  - this requires the user to provide a semicolon-separated list of Face pairs
- `COMBINE` — Bodys (of the same type) since Mark into next higher type entity
  - SheetBodys are combined into a SolidBody
  - WireBodys are combined into a SheetBody (only if the WireBodys were co-planar)

Other Boolean-like Operations (2)

- `EXTRACT` — a lower type entity
  - If Body is a SolidBody and `index > 0`
    - create SheetBody from `+index`'th Face
  - Elseif Body is a SolidBody and `index < 0`
    - create WireBody from `-index`'th Edge
  - Elseif Body is SolidBody and `index = 0`
    - create SheetBody from outer Shell of Body
  - Elseif Body is SheetBody and `index > 0`
    - create SheetBody from `+index`'th Face
  - Elseif Body is SheetBody and `index < 0`
    - create WireBody from `-index`'th Edge
  - Elseif Body is SheetBody and `index = 0`
    - create WireBody from outer Loop (currently not implemented)

Transformations (1)

- Pops one Group off the stack
- Pushes the transformed Group onto the stack
- Supported transformations include:
  - `TRANSLATE` — move the entity to another location
  - `ROTATEX, ROTATEY, ROTATEZ` — rotate the entity around an axis that is parallel to the x-, y-, or z-axis
  - `SCALE` — change the size of the entity
  - `MIRROR` — create the mirror image of the entity across an arbitrary plane (specified by a unit normal and distance from the origin)
  - `APPLYCSYS` — apply transformation given by a CSYSTEM
  - ...

Transformations (2)

- ...
  - `REORDER` — change the order in which the Edges are listed in a Sketch
    - this does NOT change the geometry
    - sometimes useful before calling `RULE` or `BLEND`
    - often the `reorder` argument to `RULE` or `BLEND` is preferrable

Other Primitives

- Other primitives include:
  - `IMPORT` — create a Body by reading from an external file:
    - `.step`, `.stp`, and `.STEP`
    - `.iges` and `.igs`
    - `.egads`
  - `UDPRIM` — execute a user-defined primitive
    - arguments are provided in keyword-value pairs
    - arguments can be "pre-loaded" with `UDPARG` statements
    - keywords are defined by the writer of the UDP
  - `RESTORE` — Body that was previously built (during the current build process) and "stored" can be restored
    - all attributes are kept during the `STORE` and `RESTORE`

Differences Between UDPs and UDFs

- Users can add their own user-defined primitives (UDPs)
  - createa 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 two Bodys from the stack

Calling a UDP

- 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

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

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

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
```

UDPs Shipped with ESP (1)

- `bezier` — generate a Bezier WireBody, SheetBody, or SolidBody from a input file
- `biconvex` — generate a biconvex airfoil
- `box` — generate a (rectangular) WireBody, SheetBody, or SolidBody centered at the origin (with possibly-rounded corners)
- `csm` — call `OpenCSM` recursively to read a `.csm` file and create a Body
- `ellipse` — generate an ellipse centered at the origin (try to use the `supell` UDP instead)
- `fitcurve` — fit a Bspline curve to a set of points
- `freeform` — generate a freeform WireBody, SheetBody, or SolidBody from an input file
- `hex` — create a general hexahedron from its corners segments

UDPs Shipped with ESP (2)

- `import` — read a Body out of a `.step` file
- `kulfan` — generate a Kulfan airfoil
- `naca` — generate a NACA 4-series airfoil or camberline
- `naca456` — generate a NACA 4-, 5-, or 6-series airfoil
- `nurbbody` — generate a Body from a seried of NURBS
- `parsec` — generate a Parsec airfoil by either specifying Sobieski's parameters or spline parameters
- `pod` — generates a VSP-like pod
- `poly` — generate a general polyhedron, polygon, line, or point
- `radwaf` – generate a radial waffle, which is useful for creating fuselage structures

UDPs Shipped with ESP (3)

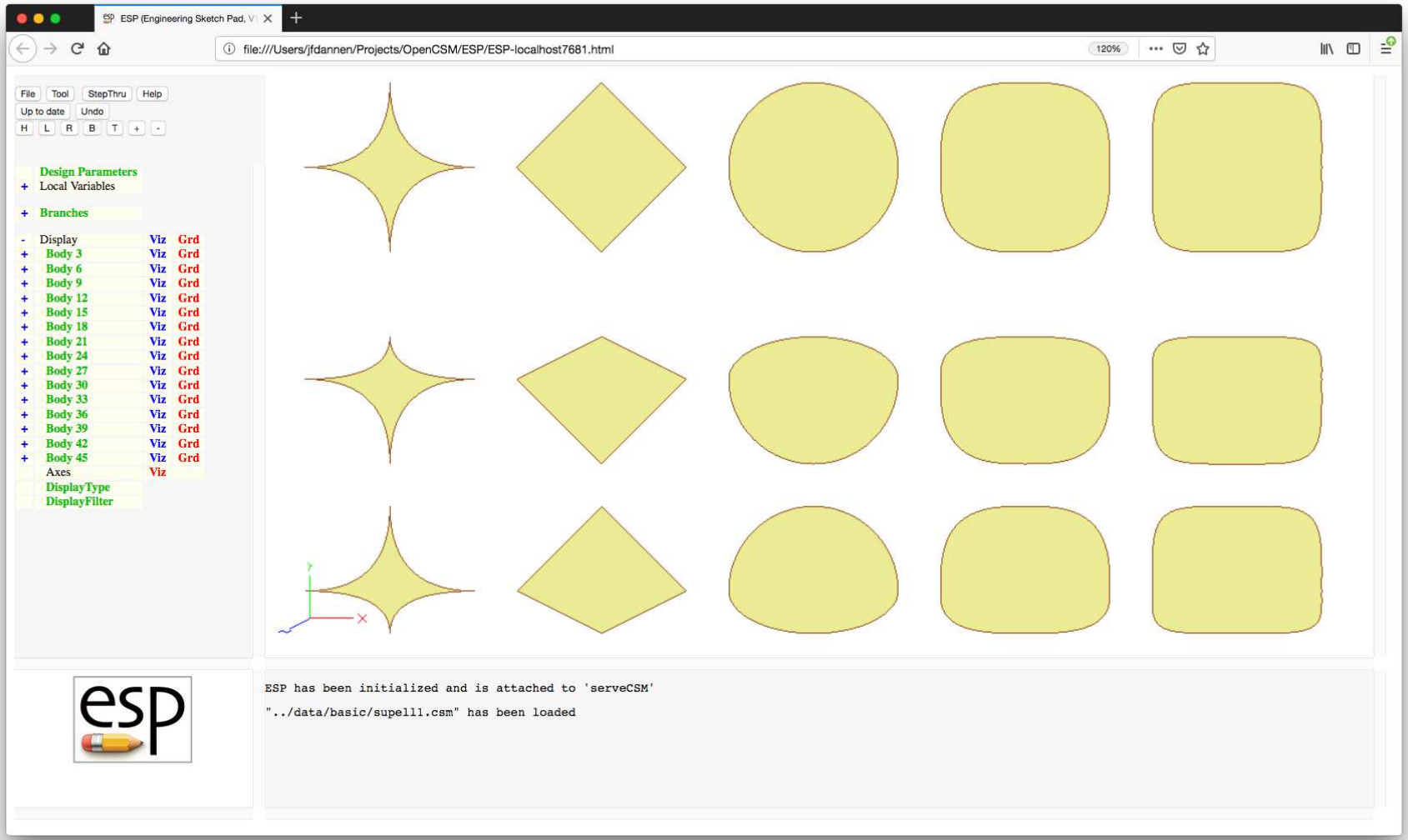
- `sample` — used as an example for users who want to create their own UDP
- `sew` — sew Faces in a `step` file into a SolidBody
- `stag` — simple turbomachinery airfoil generator
- `supell` — generate a 4-quadrant super-ellipse
- `waffle` — generate a waffle by extruding a 2D group of segments

Creating NACA Airfoils

```
# naca

UDPRIM naca thickness 0.00 camber 0.04
TRANSLATE -2 0 0

UDPRIM naca thickness 0.12 camber 0.00

UDPRIM naca thickness 0.12 camber 0.04
TRANSLATE +2 0 0

END
```

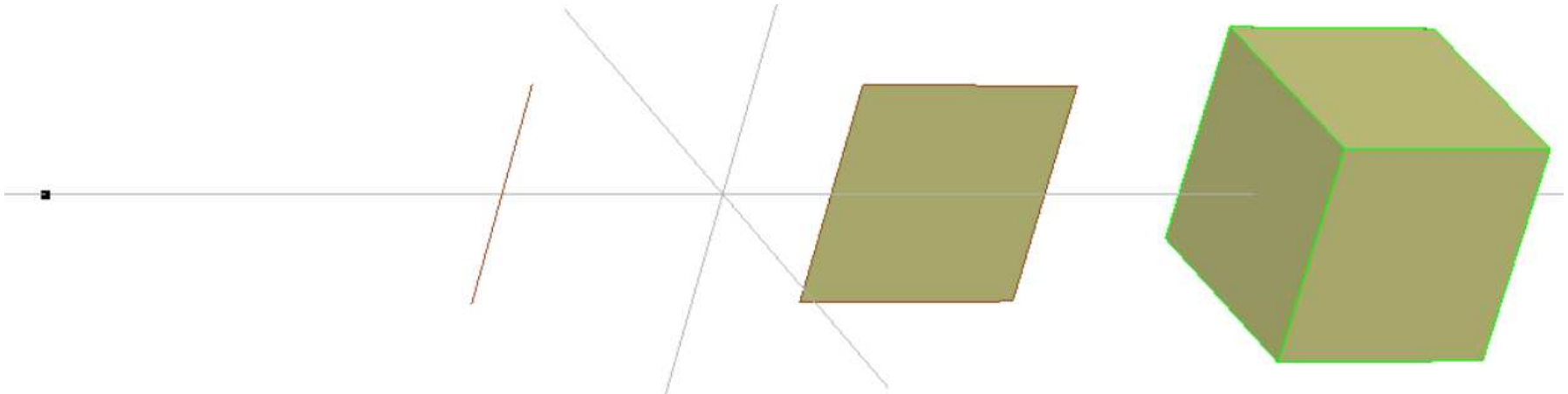
Creating Super-ellipses

Generated with `$ESP_ROOT/data/basic/supell1.csm`



Creating Simple Bodys

```
# simple

POINT     -3 0 0

UDPRIM box  dy 1.0
TRANSLATE -1 0 0

UDPRIM box  dx 1.0  dy 1.0
TRANSLATE +1 0 0

UDPRIM box  dx 1.0  dy 1.0  dz 1.0
TRANSLATE +3 0 0

END
```



UDFs Shipped with ESP

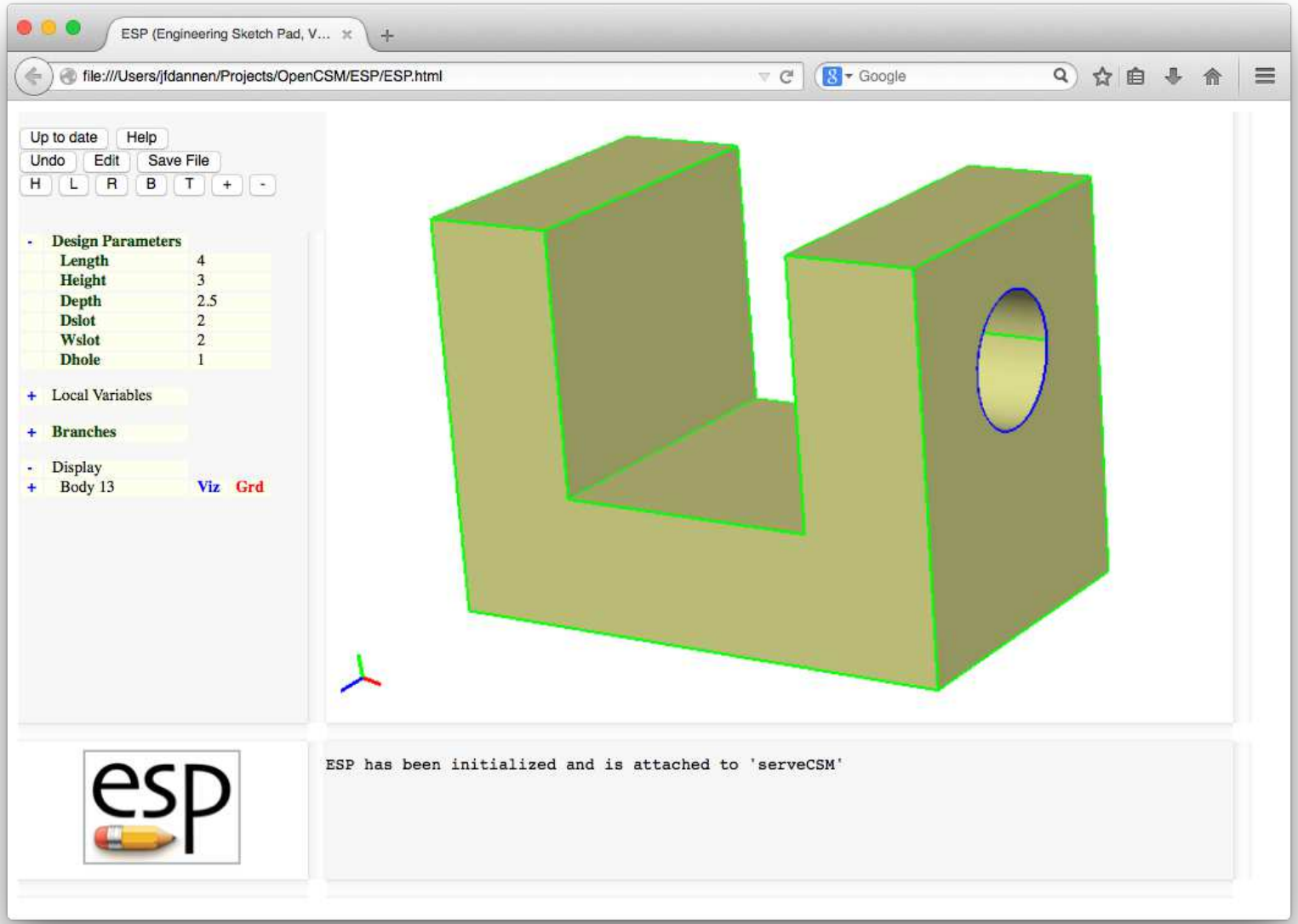
- `createBEM` — create a NASTRAN-type built-up-element (BEM) file from Body on Stack
- `createPoly` — create a TETGEN `.poly` file between the two Bodys on the top of the Stack
- `editAttr` — edit the Attributes for the Body on the top of the stack

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

Hands-on Activities

- U-shaped bracket
- Simple block
- Files in `$ESP_ROOT/training/session1.5` will get you started

U-shaped Bracket with Hole (1)



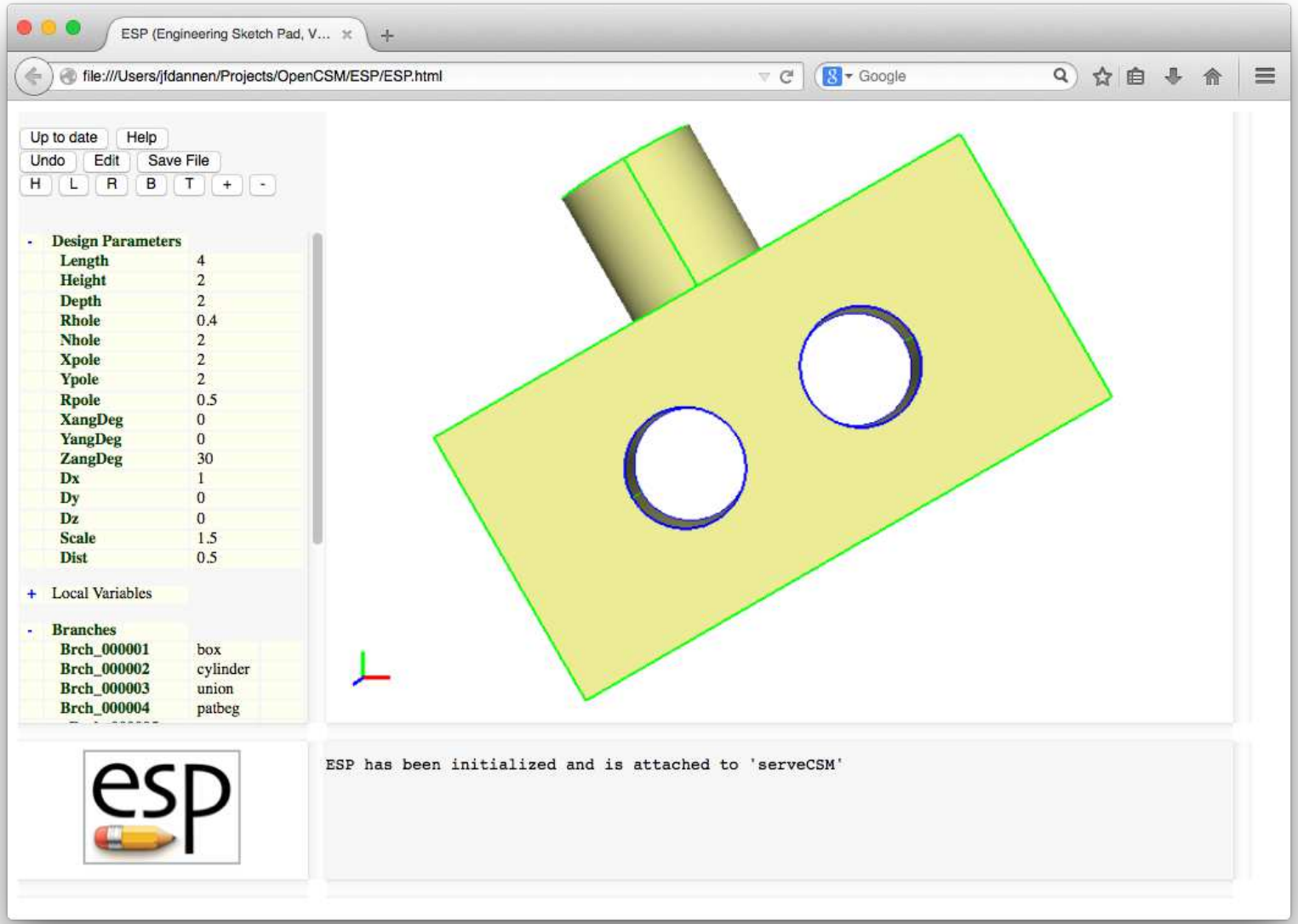
U-shaped Bracket with Hole (2)

Length	length in (X-direction)	4.00
Height	height of the two legs (Y-direction)	3.00
Depth	depth (in Z-direction)	2.50
Dslot	depth of slot (in Y-direction)	2.00
Wslot	width of slot (in X-direction)	2.00
	slot is centered in X-direction	
Dhole	diameter of hole	1.00
	hole is centered in Z-direction	
	center of hole is down Dhole from top	

U-shaped Bracket with Hole (3)

- Can you think about two different ways of creating the bracket? What are the consequences?
- What happens when you change a Design Parameter?
- What happens if you make `Dhole` large?
- What happens if you add a `FILLET` after subtracting the hole?

Simple Block (1)



Simple Block (2)

Box		
Length	length of box	4.0
Height	height of box	2.0
Depth	depth of box	2.0
	anchored at $X = Z = 0$	
	centered at $Y = 0$	
Holes		
Rhole	radii of the holes	0.4
Nhole	number of holes	2
	holes are equally spaced	
Pole		
Xpole	X-location of top of pole	2.0
Ypole	Y-location of top of pole	2.0
Rpole	radius of pole	0.5

Simple Block (3)

Rotation about origin		
XangDeg	X rotation (deg)	0.
YangDeg	Y rotation (deg)	0.
ZangDeg	Z rotation (deg)	30.
Translation		
Dx		1.0
Dy		0.0
Dz		0.0
Scaling		
Scale	overall scaling factor	1.5

Simple Block (4)

- What is the sensitivity to each Design Parameter?
- What is the sensitivity if you change two Design Parameters at the same time?