Engineering Sketch Pad (ESP)

User Training Session 2
Solids Fundamentals (1)

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

updated for v1.28

Overview

- Build Stack
- Types of Bodys
- Primitive Bodys
- Boolean Operations
- Transformations
- Homework Exercises

Build Stack

- `OpenCSM` uses the idea of a "stack" to keep track of Bodys
- A stack is a general first-in-last-out data structure that has the following characteristics:
  - a stack starts off being empty
  - an item (Body) can be "pushed" onto the top of the stack, causing the stack to grow in size
  - an item (Body) can be "popped" off the top of the stack, causing the stack to shrink in size
- The stack can contain Bodys or Marks

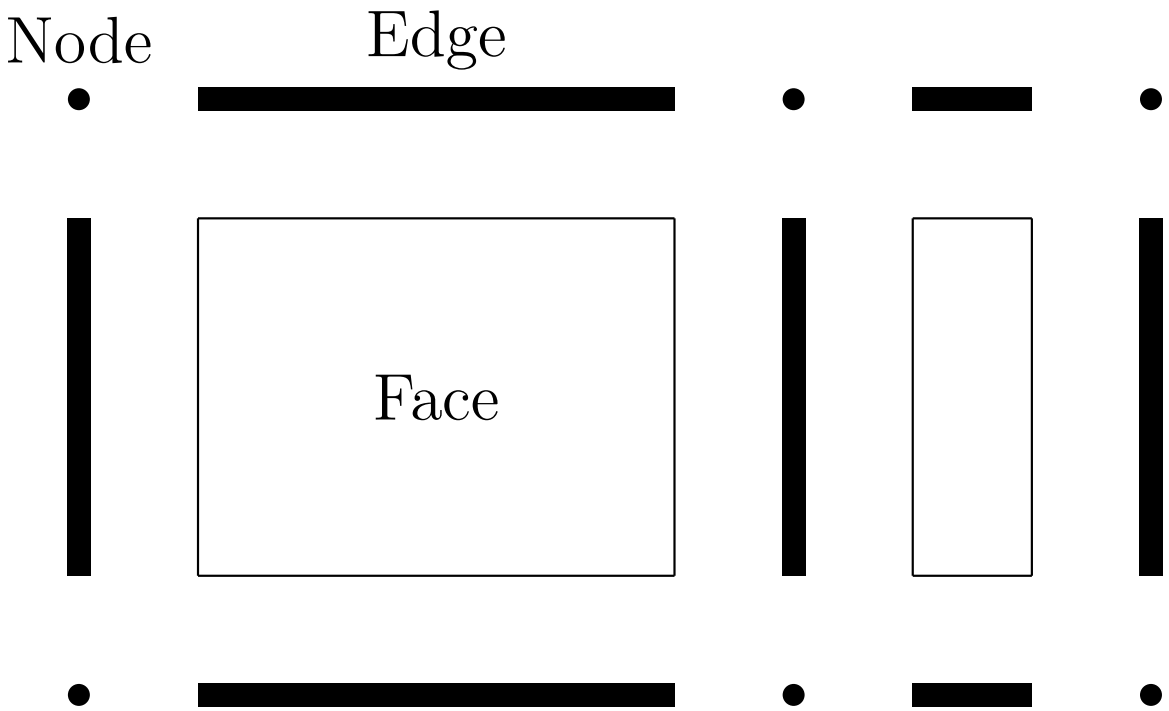
Stack Example

Operation	Stack contents after operation	Results
Start	(empty)	(none)
Push A	A	(none)
Push B	A B	(none)
Push C	A B C	(none)
Pop	A B	C
Push D	A B D	(none)
Pop	A B	D
Pop	A	B

Note: if these were Body(s), Body A would be visible in `ESP` after the build process

BRep Nomenclature (1)

"BRep" means Boundary Representation



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, $t_{\min} \leq t \leq t_{\max}$
  - 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 the length is zero)

BRep Nomenclature (3)

- Loop
  - free standing collection of one or more connected Edges with associated senses (forward or reverse)
  - 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 or 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

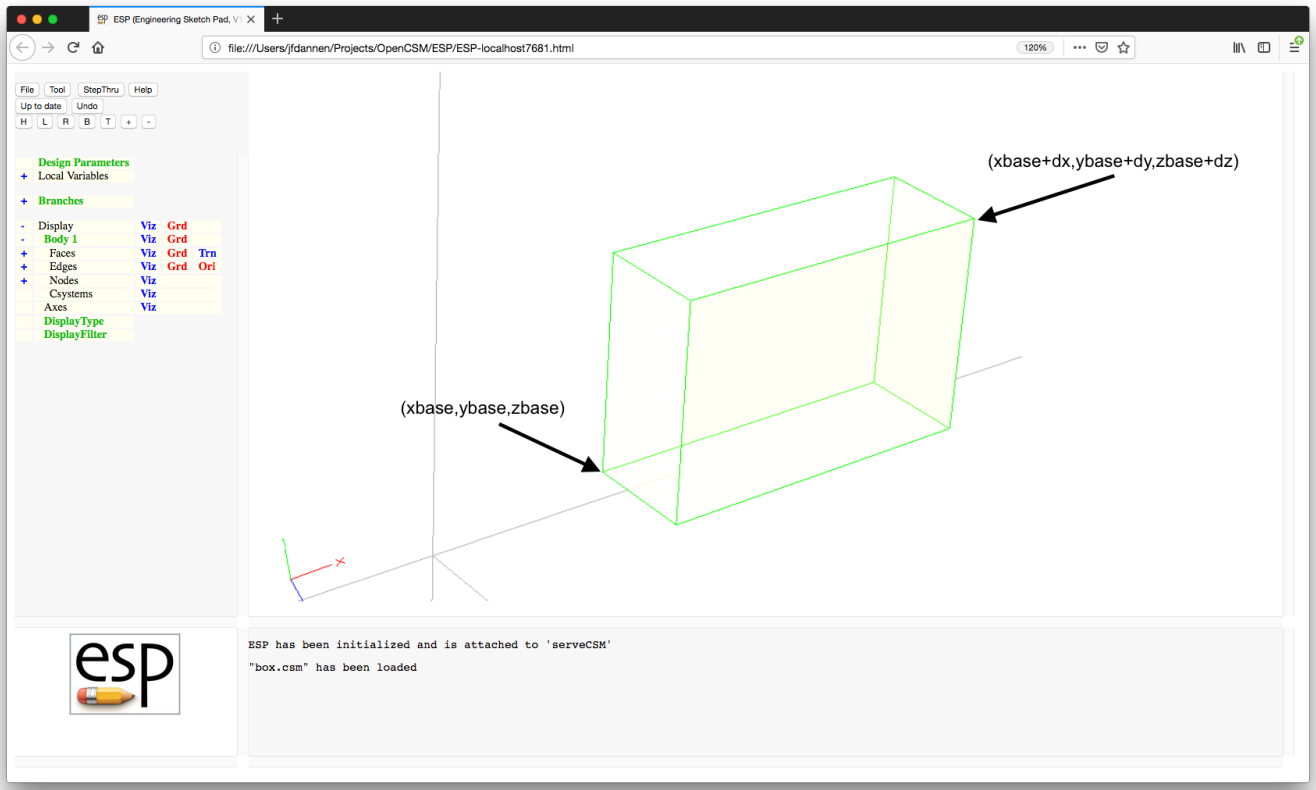
- SolidBody
  - Body that has an inside and outside
  - bounded by a Shell
  - may contain one or more voids, each of which is defined by a Shell
- SheetBody
  - a single Shell that can be either non-manifold (open) or manifold (closed)
- WireBody
  - a single Loop, with possible attached Edges (if non-manifold)
- NodeBody
  - a single Node (represented internally as a degenerate WireBody)

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 (1)

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



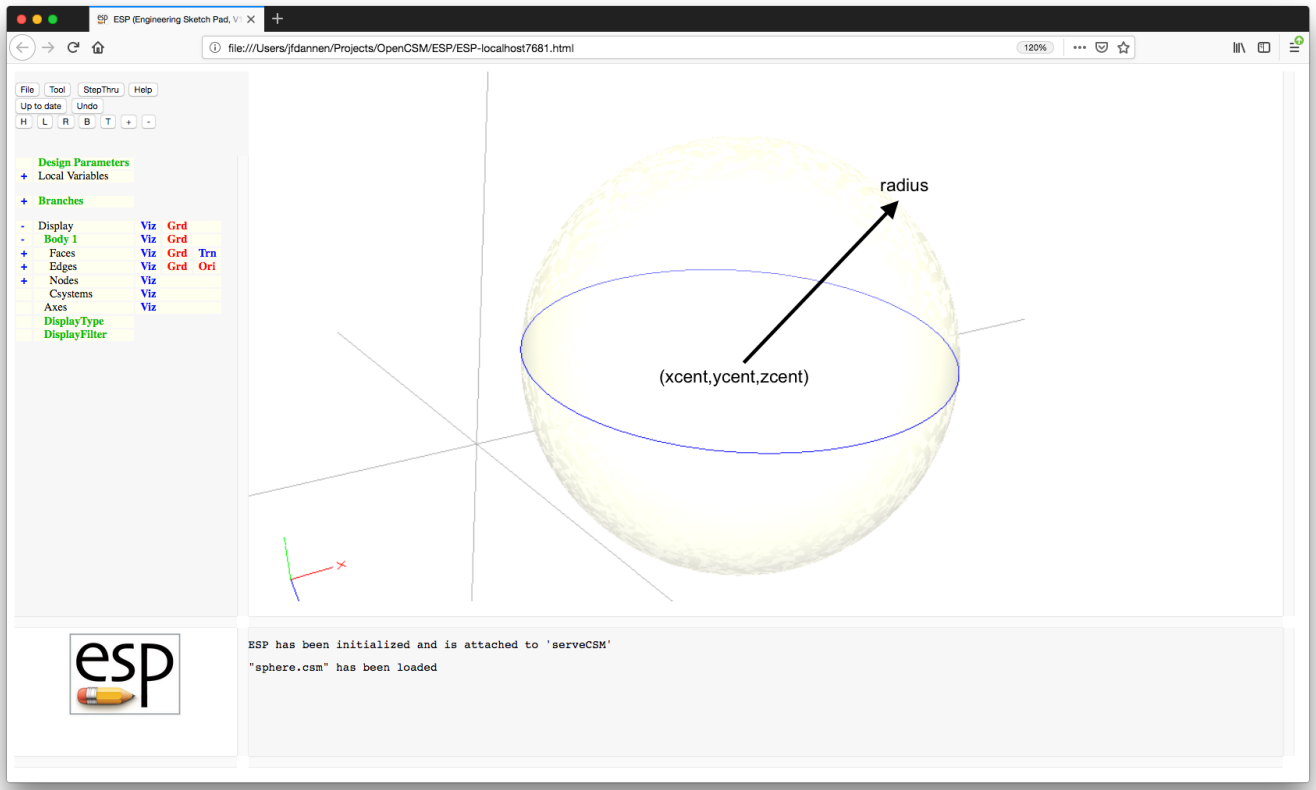
- Face-order:
  - 1: x min
  - 2: x max
  - 3: y min
  - 4: y max
  - 5: z min
  - 6: z max

Standard Primitive — BOX (2)

- Special rules for a BOX
  - if `dx=0`, or `dy=0`, or `dz=0`, a SheetBody is created
  - if `dx=dy=0`, or `dy=dz=0`, or `dz=dx=0`, a WireBody is created
  - if `dx=dy=dz=0`, a NodeBody is created

Standard Primitive — SPHERE

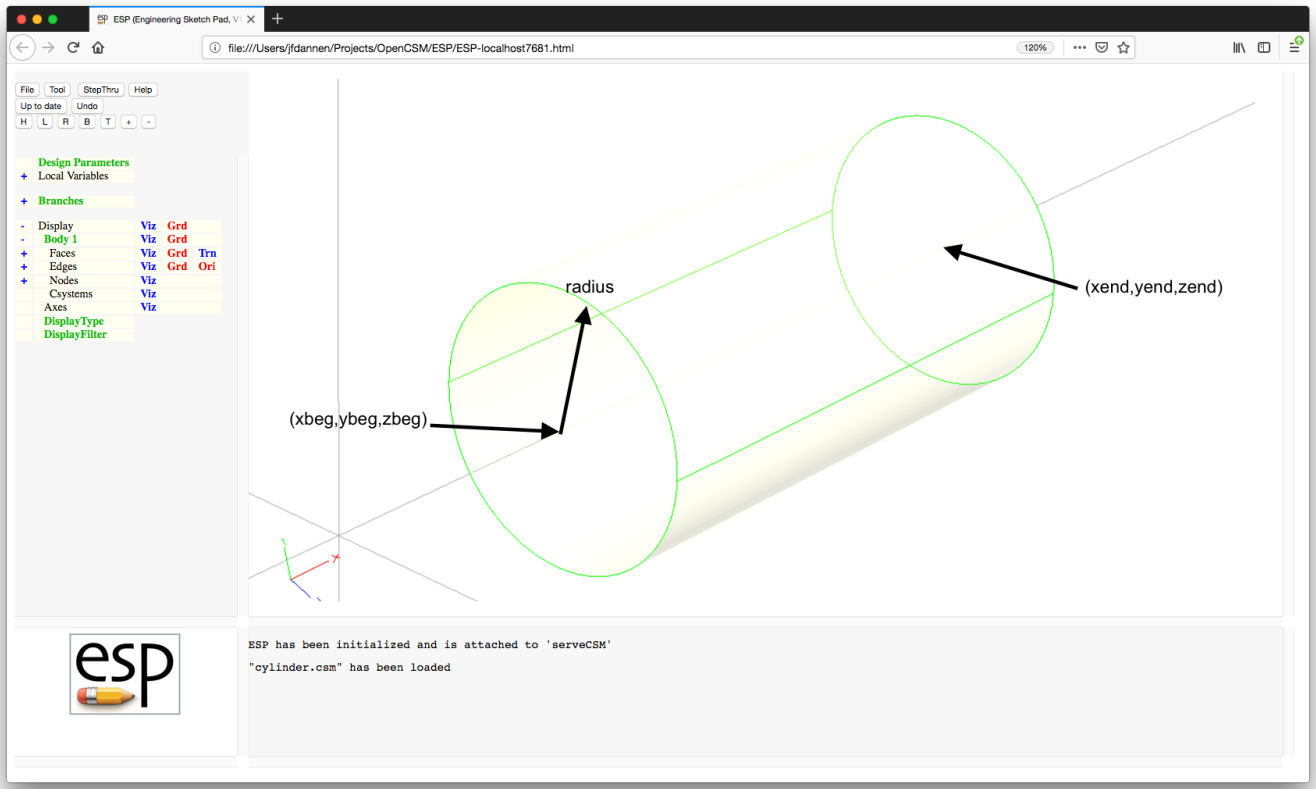
```
SPHERE xcent ycent zcent radius
```



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

Standard Primitive — CYLINDER

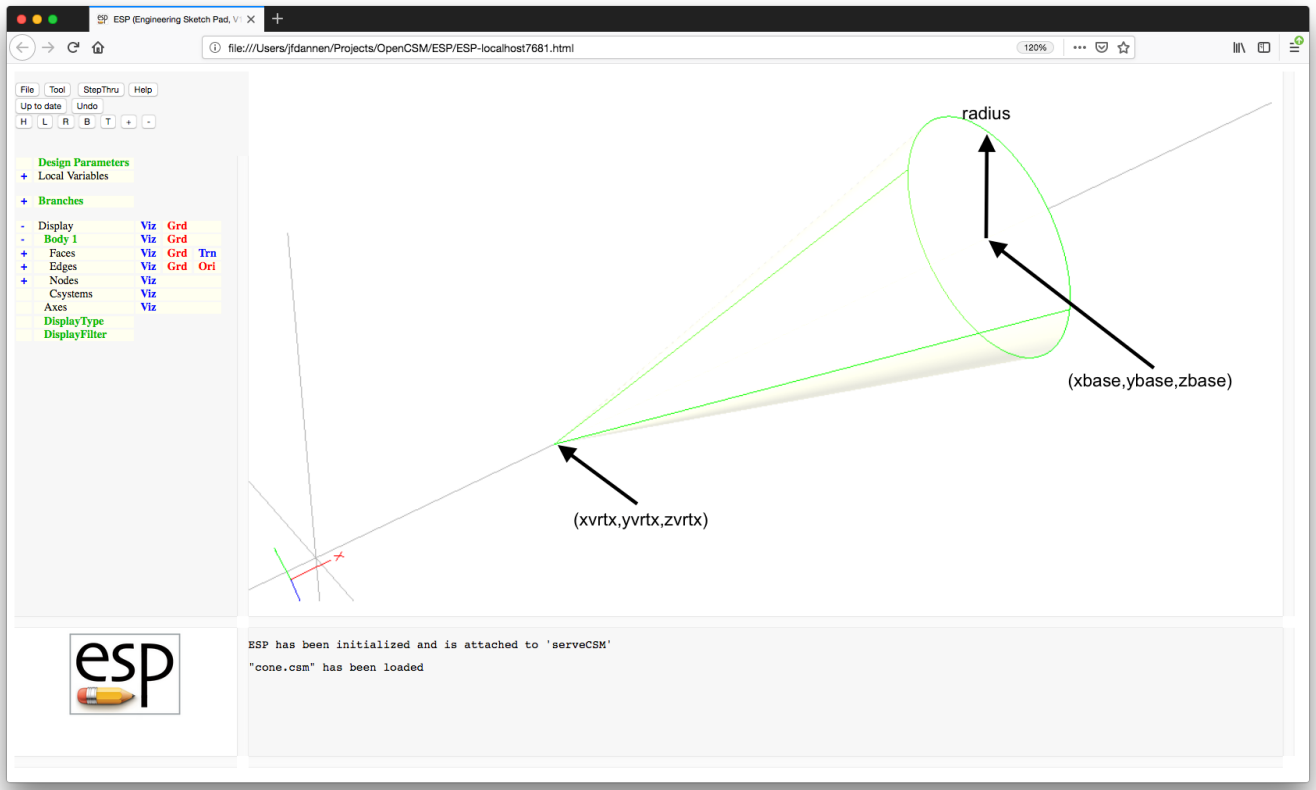
`CYLINDER xbeg ybeg zbeg xend yend zend radius`



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

Standard Primitive — CONE

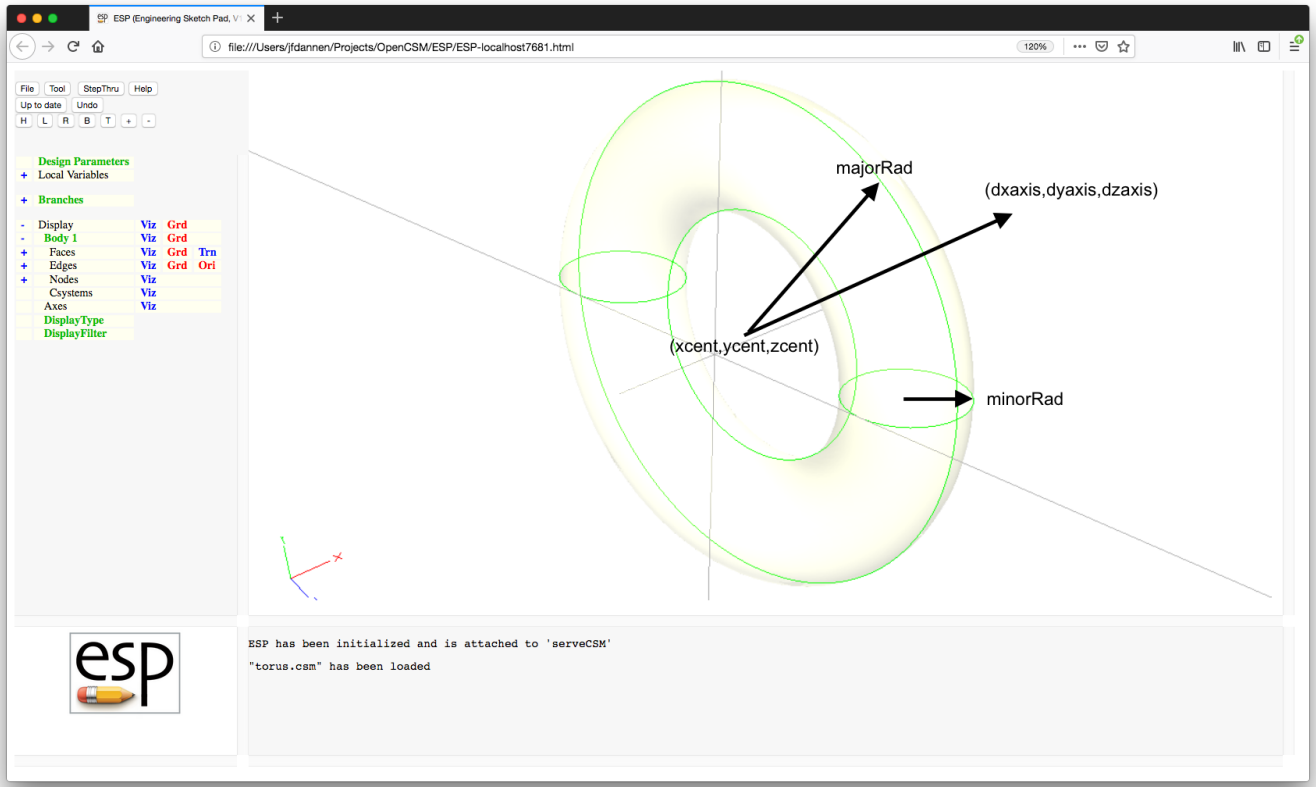
`CONE xvrtx yvrtx zvrtx xbase ybase zbase radius`



- 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

Boolean Operations (1)

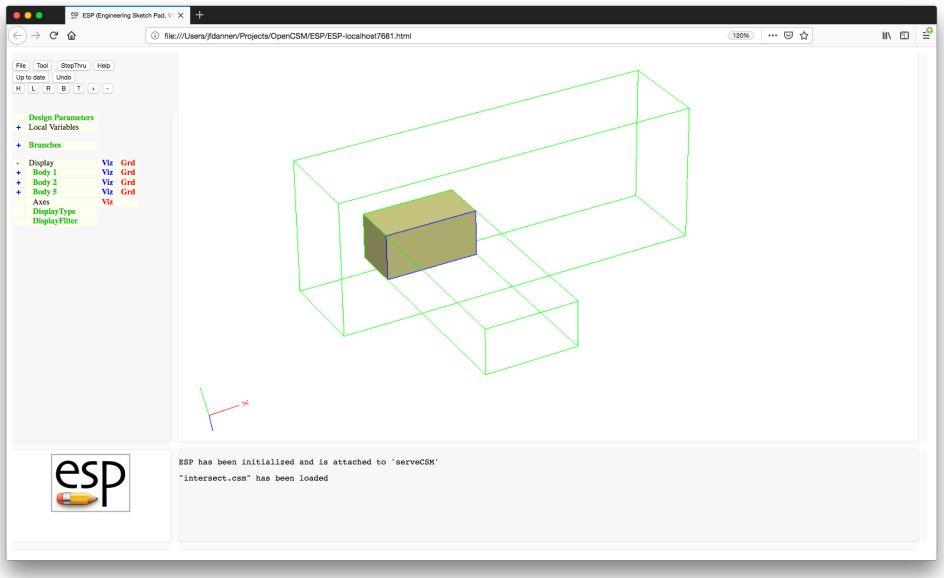
- Pops two* Bodys off the stack
- Pushes the result onto the stack
- Supported Booleans include:
  - `INTERSECT` — find the "common" parts of the input Bodys
    - if more than one Body results, order them based upon the `$order` argument and keep the `index`'th one
  - `SUBTRACT` — remove one Body from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

Boolean Operations (2)

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

Boolean Operation — INTERSECT

Note: Original Bodys and result of INTERSECT are shown



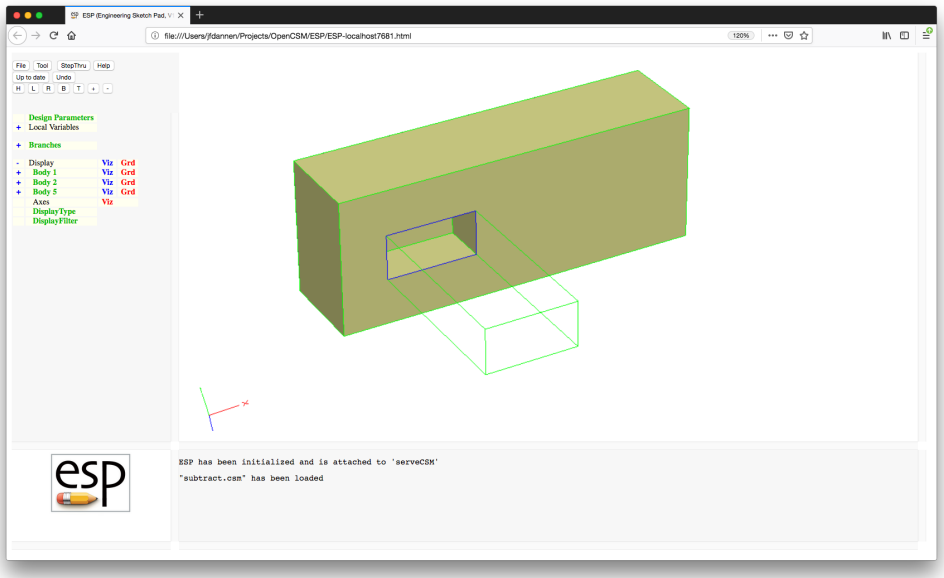

```
# intersect

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

Note: Original Bodys and result of SUBTRACT are shown




```
# 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)

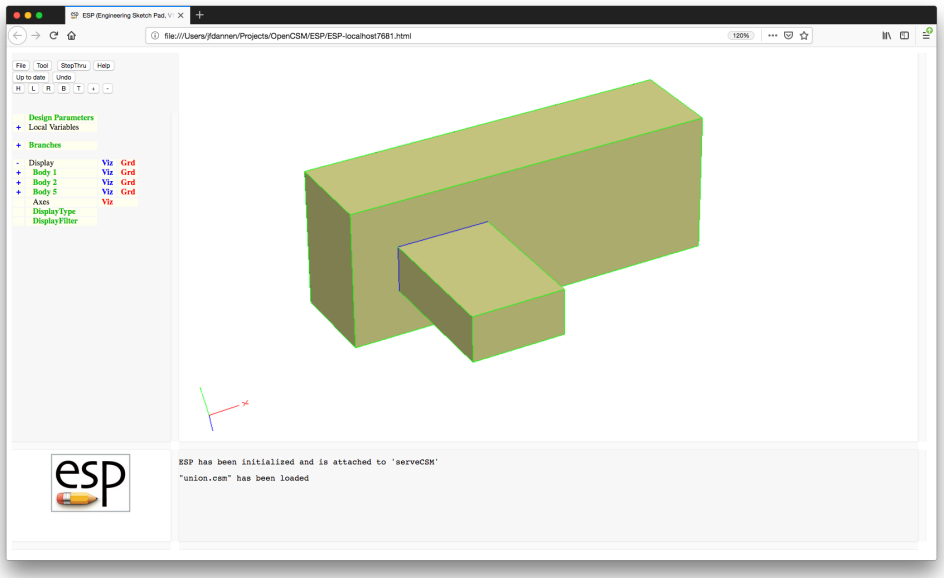
- Call the last Body on the Stack Body2 and the next-to-last Body Body1
- If Body1 and Body2 are both SolidBodys
  - create a SolidBody that is the part of Body1 that is outside Body2
- If Body1 is a SheetBody and Body2 is a Solid Body
  - create a SheetBody that is the part of Body1 that is outside 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 are created and Edges are created where the SheetBody completely cuts a Face)
- If Body1 and Body2 are non-coplanar SheetBodys
  - create a SheetBody that is the same shape as Body1, but which is scored with the intersection of Body2 (ie, new Nodes are created and Edges are created where the SheetBody completely cuts a Face)
- If Body1 and Body2 are coplanar SheetBodys
  - create a SheetBodys that is Body1 with Body2 cut out of it
- Other combinations of input Bodys are not allowed

Boolean Operation — UNION

Note: Original Bodys and result of UNION are shown




```
# 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 point prescribed in the `trimList`

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
- `ELEVATE` — Body into next higher type entity
  - SheetBodys are converted into a SolidBody if closed;
  - WireBodys are combined into a SheetBody (only if the WireBodys were co-planar). Also, if the WireBody is planar, but open, it is closed first.

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
  - Note: more than one entity can be extracted if the argument is a semicolon-separated list

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 preferable

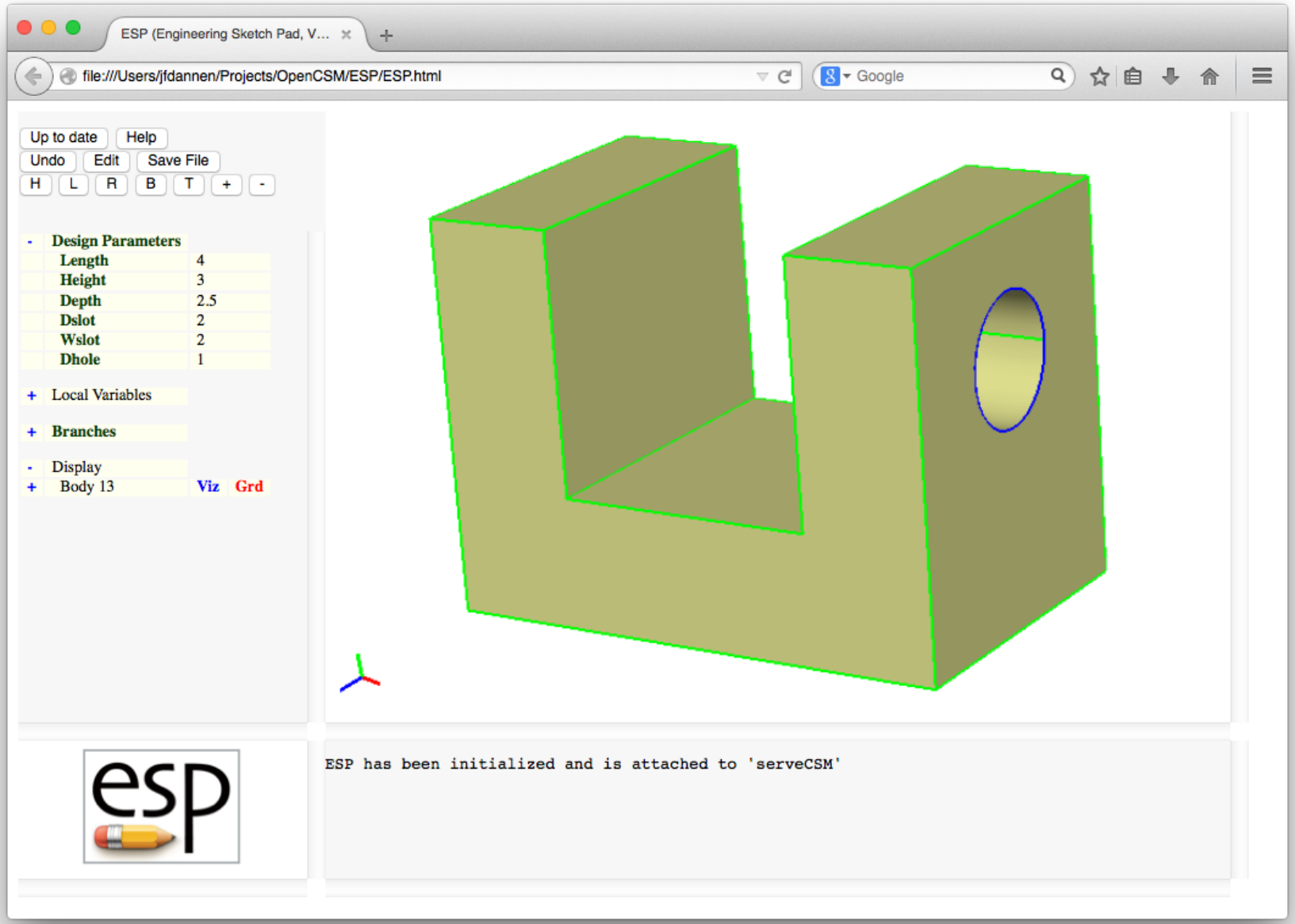
Other Primitives

- Other primitives include:
  - `IMPORT` — create a Body by reading from an external file:
    - `.step`, `.stp`, `.STEP`, and `.STP`
    - `.iges`, `.igs`, `.IGES`, and `.IGS`
    - `.egads` and `.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`
    - `RESTORE name index=0` restores Body from specified storage
    - `RESTORE .` duplicates the Body on the top of the Stack
    - `RESTORE ..` duplicates the Bodys on the Stack back to the Mark (including the Mark)
    - `RESTORE ...` duplicates all Bodys on the Stack

Homework exercises

- U-shaped bracket
- Simple block
- Files in `$ESP_ROOT/training/ESP/exercises/session02` will get you started
  - the files contain the various Design Parameters (`DESPMTR`s) in the tables on the following pages
  - for now, build up your configuration in `ESP` by pressing `Branches` in the Tree Window and then filling in the necessary information
  - you can always press the `Undo` button if you make a mistake
  - you will need to add `BOX`, `CYLINDER`, `UNION`, `SUBTRACT`, `TRANSLATE`, `SCALE`, `ROTATEX`, `ROTATEY`, and `ROTATEZ` commands/statements

U-shaped Bracket with Hole (1)



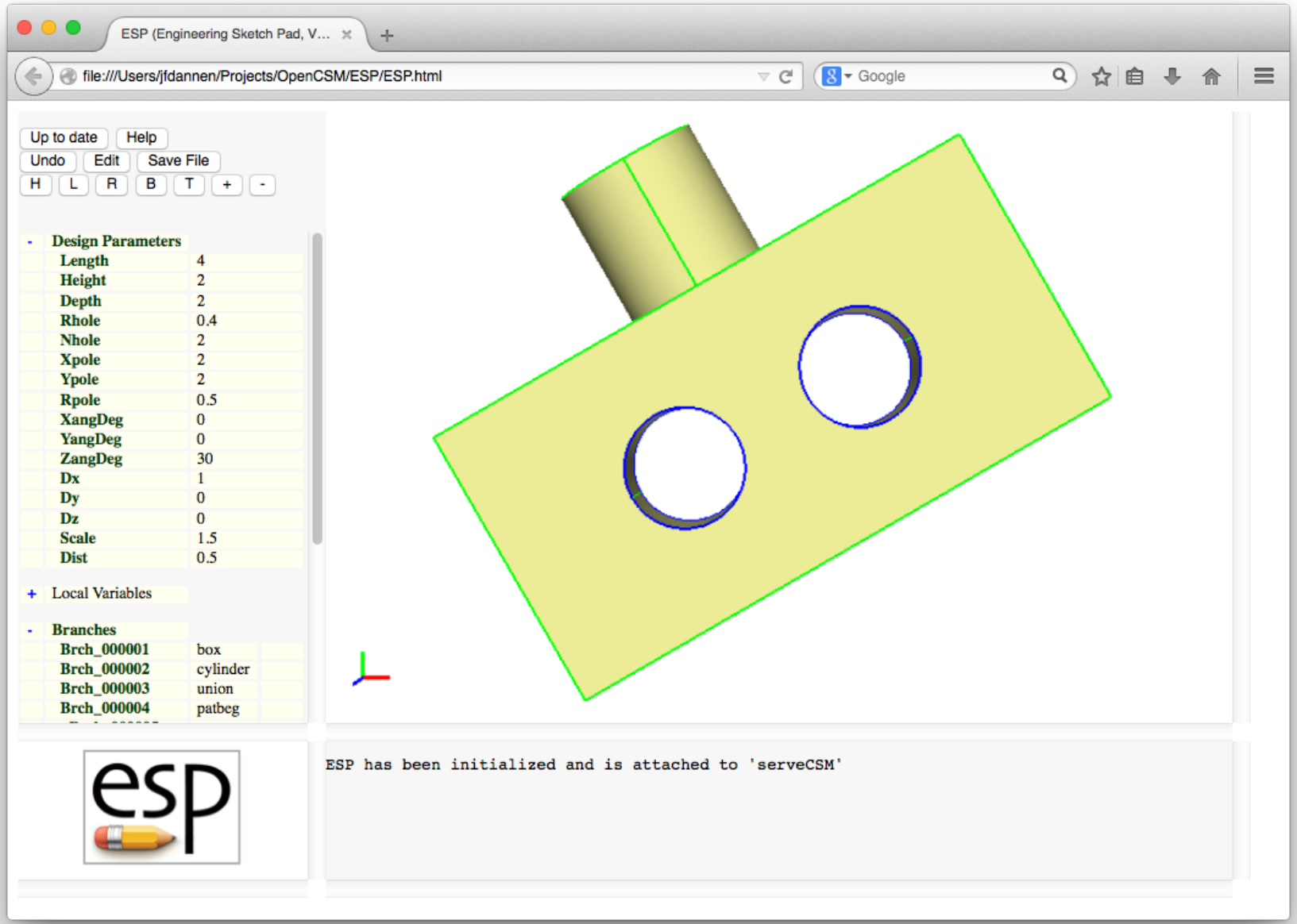
U-shaped Bracket with Hole (2)

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?

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 happens if you make the depth of the box very large?
- What is the effect of applying the various transformations in a different order?

- Don't forget to submit "muddy cards"