Engineering Sketch Pad (ESP)

Training Session 5
CSM Language (2)

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

Bob Haimes
haimes@mit.edu
Massachusetts Institute of Technology

updated for v1.18

Overview

- Manipulating the Stack
  - GROUP
  - STORE, RESTORE
- Looping
  - PATBEG, PATBREAK, PATEND
- Logic
  - IFTHEN, ELSEIF, ELSE, ENDIF
- Signal Handling
  - THROW, CATBEG, CATEND
- User-defined Components (UDCs)
  - Include-style
  - Function-style
- Homework Exercises

Manipulating the Stack (1)

- During the build process, `OpenCSM` maintains a last-in-first-out (LIFO) “Stack” that can contain Bodys and Sketches.
- The `.csm` statements are executed in a stack-like way, taking their inputs from the Stack and depositing their results onto the Stack.
- Bodys can be grouped with the `GROUP` statement
  - all the Bodys back to the Mark (or the beginning of the Stack) are put into a single Group
  - some operations, such as the transformations, `ATTRIBUTE`, `STORE`, and `DUMP` operate on all Bodys in the Group simultaneously
  - Bodys and be ungrouped by giving `GROUP` a negative argument

Manipulating the Stack (2)

- The Group on the top of the Stack can be “popped” off the stack with a `STORE` command
    - if the `name` is alpha-numeric, the Group is stored in a named storage location
    - if the `name` is a dot (.), the Group is not stored (just popped off the Stack)
    - if the `name` is two dots (..), all the Groups back to the Mark are popped off the Stack (and not stored)
    - if the `name` is three dots (...), everything is popped off the Stack

Manipulating the Stack (3)

- Groups can be read from a named storage location and "pushed" onto the Stack with the `RESTORE` command
- The `RESTORE` command is considered a primitive, so its Attributes are put on all the Bodys and all their Faces

Patterns (1)

- Patterns are like “for” or “do” loops
    - the Branches between the PATBEG and PATEND are executed a known number of times
    - at the beginning of each “instance”, the pattern number is incremented (from 1 to the number of copies)
    - one can break out of the pattern early with a PATBREAK statement
    - patterns can be nested within other patterns

Patterns (2)

- Example pattern (indentation optional):

```
PATBEG     i    3
   SET     j    i-1
   BOX     j    0  0  1  1  1
   ROTATEX j*10 0  0
PATEND
```

- is the same as:

```
BOX     0  0  0  1  1  1
ROTATEX 0  0  0

BOX     1  0  0  1  1  1
ROTATEX 10 0  0

BOX     2  0  0  1  1  1
ROTATEX 20 0  0
```

If/then (1)

- If/then constructs are used to make a choice within a `.csm` script
  - start with `IFTHEN` statement
  - has zero or more `ELSEIF` statements
  - has zero or one `ELSE` statement
  - has exactly one `ENDIF` statement
- The `IFTHEN` and `ELSEIF` statements have arguments, some of which can be specified in lowercase or UPPERCASE
  - `val1` — an expression
  - `op1` — can be `lt`, `le`, `eq`, `ge`, `gt`, `ne`, `LT`, ...
  - `val2` — an expression
  - `op2` — can be `or`, `xor`, `and`, `OR`, ... (defaults to `and`)
  - `val3` — an expression (defaults to `0`)
  - `op3` — can be `lt`, `le`, `eq`, `ge`, `gt`, `ne`, `LT`, or ... (defaults to `eq`)
  - `val4` — an expression (defaults to `0`)

If/then (2)

- Example (indentation optional):

```
IFTHEN    a  eq  4  or  b  ne  2
   BOX    0  0   0  1   1  1
ELSEIF    c  eq  sqrt(9)
   BOX    2  2   2  2   2  2
ELSE
   BOX    3  3   3  3   3  3
ENDIF
```

Throw/catch (1)

- Throw/catch constructs are used to generate and react to signals (errors)
- Signals can be generated by
  - executing a `THROW` command
  - a run-time error encountered elsewhere (see “help” for more info)
- When a signal is generated, all Branches are skipped until a matching `CATBEG` statement is encountered
  - the signal is cancelled
  - processing continues at the statement following the `CATBEG`
- If a `CATBEG` statement is encountered when there is no pending signal (or the pending signal does not match the `CATBEG`)
  - all Branches up to, and including the matching `CATEND` statement, are skipped

Throw/catch (2)

```
 1: BOX      0 0 0 1 1 1

 2: THROW    -99

 3: SPHERE   0 0 0 1

 4: CATBEG   -98
 5:    SPHERE   0 0 0 2
 6: CATEND

 7: SPHERE   0 0 0 3

 8: CATBEG   -99
 9:    BOX         1 0 0 1 1 1
10: CATEND

11: CATBEG   -99
12:    SPHERE  0 0 0 4
13: CATEND

14: END
```

- `BOX` in line 1 is generated
- `SPHERE` in line 3 is skipped (since there is an active signal)
- `CATBEG/CATEND` in lines 4–6 are skipped (since they do not match `-99`)
- `SPHERE` in line 7 is skipped
- `BOX` in line 9 is generated
- `CATBEG/CATEND` in lines 11–13 are skipped (since the signal was cancelled when it was caught in line 8)

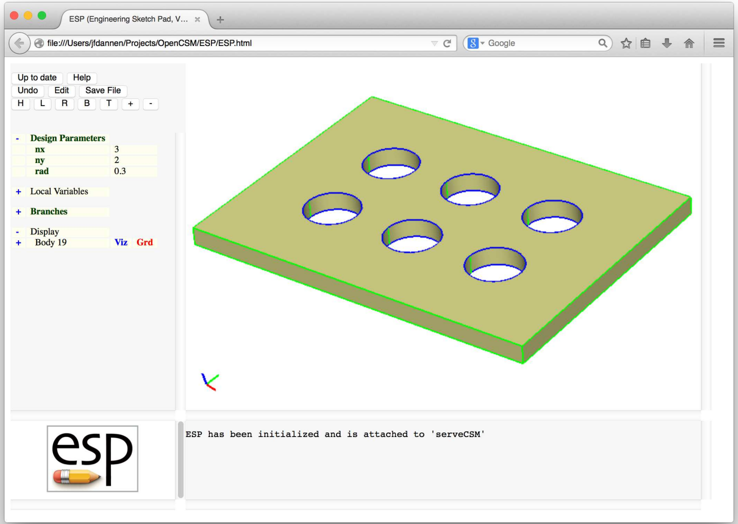
Special Note on Programming Blocks

- Programming Blocks are delineated by
  - `PATBEG` and `PATEND`
  - `IFTHEN`, `ELSEIF`, `ELSE`, and `ENDIF`
  - `SOLBEG` and `SOLEND`
  - `CATBEG` and `CATEND`
- Any programming Block can be nested fully within any other programming Block (up to 10 levels deep)

Homework Exercises

- Rectangular plate with holes
- Round plate with holes
- Determine if two Bodys overlap
- Files in `$ESP_ROOT/training/ESP/data/session05` will get you started

Rectangular Plate with Holes (1)



nx	3
ny	2
rad	0.3

```
ESP has been initialized and is attached to 'serveCSM'
```

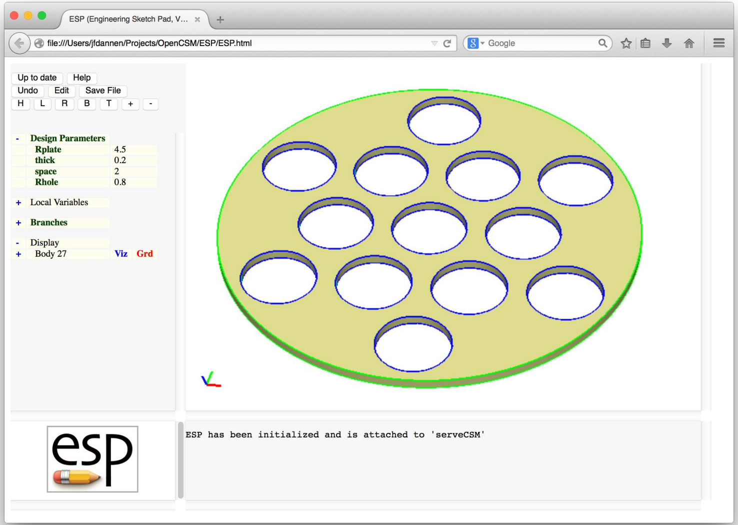
Rectangular Plate with Holes (2)

nx	number of holes in X-direction	3.00
ny	number of holes in Y-direction	2.00
rad	radius of each hole	0.30
	distance between hole centers	1.00

Rectangular Plate with Holes (3)

- Can you make a single hole in the center of the plate?
- Can you change your solution to have the holes spaced so that they fill the plate?
- What if you make the radius of the hole too big?

Round Plate with Holes (1)



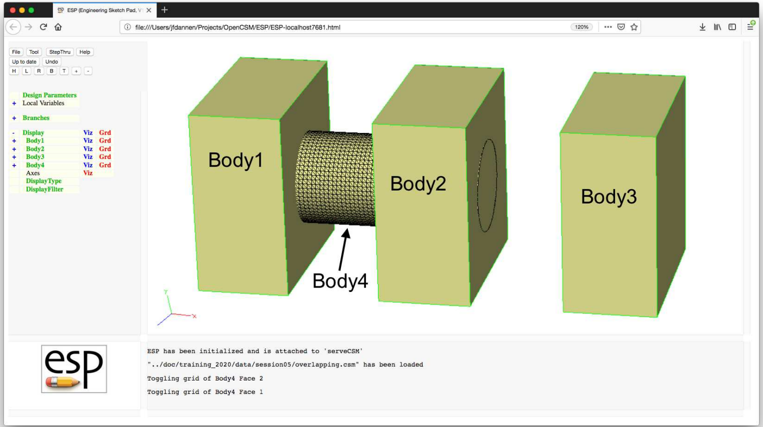
Design Parameters	
Rplate	4.5
thick	0.2
space	2
Rhole	0.8

```
ESP has been initialized and is attached to 'serveCSM'
```

Round Plate with Holes (2)

Rplate	radius or plate	4.50
thick	thickness of plate	0.20
space	distance between hole centers	2.00
Rhole	radius of holes	0.80
	number of holes selected automatically	

Overlapping Bodys (1)



```
ESP has been initialized and is attached to 'serveCSM'
"../doc/training_2020/data/session05/overlapping.csm" has been loaded
Toggling grid of Body4 Face 2
Toggling grid of Body4 Face 1
```

Overlapping Bodys (2)

- Write `.csm` file to:
  - set `overlap1` to 1 if Bodys 1 and 4 overlap, otherwise set it to 0
  - set `overlap2` to 1 if Bodys 2 and 4 overlap, otherwise set it to 0
  - set `overlap3` to 1 if Bodys 3 and 4 overlap, otherwise set it to 0
- Try to use a pattern to do this compactly