

# Engineering Sketch Pad (ESP) Training

## Session 3: Solids Fundamentals

John F. Dannenhoffer, III

Syracuse University

Bob Haimes

Massachusetts Institute of Technology

Revised for v1.08





# Overview

- Constructive solid modeling process
- Feature Tree
- Branch types
  - primitives
  - grown Bodies (from Sketches)
  - Boolean combinations
  - transformations
  - applied features
  - miscellaneous
- Hands-on exercises
  - L-shaped bracket with hole
  - bolt
  - simple wing



# Constructive Solid Modeling Process

- Configurations start with the generation of primitive Bodys
  - standard primitives
  - grown primitives (from sketches)
  - user-defined primitives (UDPs)
- Bodys can be combined
  - Booleans
  - other
- Bodys can be modified
  - transformations
  - applications

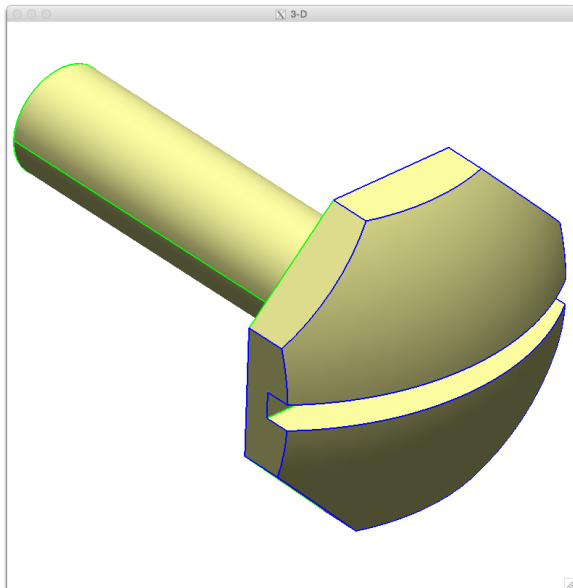


# Feature Tree (1)

- The Feature Tree can be thought of as a (binary) tree that contains the sequence of operations needed to build a configuration
- The Branches of the Feature Tree are defined by statements that specify:
  - Branch type
  - arguments (which may sometimes have default values)
- Branches are assigned default names when they are created
  - the name can be changed by the user to make the Feature Tree a bit more readable
- Branches can be attributed
  - the Branch's attributes are assigned to all Edges and Faces that are created when the Branch executes

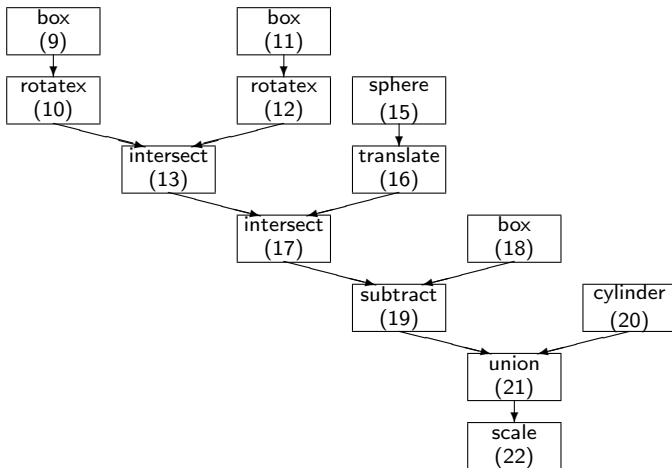


## Feature Tree (2)





## Feature Tree (3)





## Feature Tree (4)

- Branches are initially loaded into ESP by
  - reading a .csm file
  - through the ESP GUI
- When the Feature Tree is executed, it starts with an empty “stack” or entities
  - Body(s) — Solid, Sheet, or Wire
  - Sketch(es) — actually Sheet or Wire Bodys
  - Node(s) — a degenerate Sketch
  - Mark(s) — special entity when number of inputs is variable
- The Branches in a .csm file are executed in order
  - some Branch types get their inputs by “popping” entities from the stack
  - same Branches “push” their results onto the stack
- The Bodys left on the stack are returned to the user



# Feature Tree (5a)

# bolt example

# design parameters

```

1: despmtr  Thead    1.00  # thickness of head
2: despmtr  Whead    3.00  # width      of head
3: despmtr  Fhead    0.50  # fraction  of head that is flat

4: despmtr  Dslot    0.75  # depth of slot
5: despmtr  Wslot    0.25  # width of slot

6: despmtr  Lshaft   4.00  # length  of shaft
7: despmtr  Dshaft   1.00  # diameter of shaft

8: despmtr  sfact    0.50  # overall scale factor
  
```

# head

```

9: box      0      -Whead/2 -Whead/2  Thead  Whead  Whead
10: rotatex 90  0  0
11: box      0      -Whead/2 -Whead/2  Thead  Whead  Whead
12: rotatex 45  0  0
13: intersect
  
```





## Feature Tree (5b)

```
14:  set          Rhead  ( $(Whead^2/4 + (1 - Fhead)^2 * Thead^2) / (2 * Thead * (1 - Fhead))$ )

15:  sphere      0          0    0    Rhead
16:  translate   Thead-Rhead  0    0
17:  intersect

      # slot
18:  box          Thead-Dslot  -Wslot/2  -Whead  2*Thead  Wslot  2*Whead
19:  subtract

      # shaft
20:  cylinder    -Lshaft  0    0    0    0    0    Dshaft/2
21:  union

22:  scale      sfact

23:  end
```



# 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:
  - box — starting location and size
  - sphere — center and radius
  - cone — base center, vertex, and radius
  - cylinder — beginning center, ending center, and radius
  - torus — center, orientation, major- and minor-radii



## Standard Primitives (2)

- 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



## Special Rules for `udprim` and `udparg`

- Zero or more `udparg` statements may precede a `udprim` statement
- `primetype` must match in `udparg` and `udprim` statements
- Arguments are processed in order that they are encountered



# 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
```



# Grown Primitives (from Sketches)

- Pops one or more Sketches 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 Sketches back to the Mark by straight lines
    - the first and/or last Sketch can be a Node
  - blend — connect all the Sketches back to the Mark with smooth curves
    - the first and/or last Sketch can be a Node
    - at the bounding Nodes, the user can specify the radius of curvature in two orthogonal directions
  - sweep — a Sketch along a given Wire
    - this is often problematic in OpenCASCADE
  - loft — similar to blend, but with less control



# Rule, Blend, and Loft

- rule and blend require that all Sketches have the same number of Segments, ordered in the same way
  - new Faces are made by combining all the first Segments, ...
- Sketches can be automatically reordered to help eliminate twist by setting reorder to a non-zero value
- Users can manually reorder Sketches with the reorder command (applied to a Sketch)
- Reordering only changes the order of Segments, not their shapes



# Boolean Operations

- 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
- All Booleans have an optional tolerance that can be used in cases when OpenCASCADE's geometric and topological tolerances cause an error





## Special Rules for union

- If the Bodys popped off the stack are both Solid Bodys
  - if `tomark` is not zero, all Bodys back to the Mark are unioned
  - if `trimList` is a list of six semicolon-separated numbers, then the union is trimmed to only keep the part containing the `trimList`
- If the Bodys popped off the stack are both Sheet Bodys, the union produces a Sheet Body that is the combination of its input Bodys



## Special Rules for intersect (1)

- If Body1=Solid and Body2=Solid
  - create Solid that is common part of Body1 and Body2
- Elseif Body1=Solid and Body2=Sheet
  - create Sheet that is the part of Body2 that is inside Body1 (currently not implemented)
- Elseif Body1=Solid and Body2=Wire
  - create Wire that is the part of Body2 that is inside Body1 (currently not implemented)
- Elseif Body1=Sheet and Body2=Solid
  - create Sheet that is the part of Body1 that is inside Body2
- Elseif Body1=Sheet and Body2=Sheet and Bodys are co-planar
  - create Sheet that is common part of Body1 and Body2 (currently not implemented)
- ...



## Special Rules for intersect (2)

- ...
- Elseif Body1=Sheet and Body2=Sheet and Bodys are not co-planar
  - create Wire at the intersection of Body1 and Body2 (currently not implemented)
- Elseif Body1=Sheet and Body2=Wire
  - create Wire that is the part of Body2 that is inside Body1 (currently not implemented)
- Elseif Body1=Wire and Body2=Solid
  - create Wire that is the part of Body1 that is inside Body2 (currently not implemented)
- Elseif Body1=Wire and Body2=Sheet
  - create Wire that is the part of Body1 that is inside Body2 (currently not implemented)



## Special Rules for subtract (1)

- If Body1=Solid and Body2=Solid
  - create Solid that is the part of Body1 that is outside Body2
- Elseif Body1=Solid and Body2=Sheet
  - create Solid that is Body1 scribed with Edges at intersection with Body2
- Elseif Body1=Sheet and Body2=Solid
  - create Sheet that is part of Body1 that is outside Body2
- Elseif Body1=Sheet and Body2=Sheet
  - create Sheet that is Body1 scribed with Edges at intersection with Body2 (currently not implemented)
- ...



## Special Rules for subtract (2)

- ...
- Elseif Body1=Wire and Body2=Solid
  - create Wire that is part of Body1 that is outside Body2 (currently not implemented)
- Elseif Body1=Wire and Body2=Sheet
  - create Wire that is Body1 scribed with Nodes at intersection with Body2 (currently not implemented)



## Other Boolean-like Operations (1)

- join — two Bodys at a common Face
- 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
  - Sheet Bodys are combined into a Solid Body
  - Wire Bodys are combined into a Sheet Body (only if the Wires were co-planar)



## Other Boolean-like Operations (2)

- `extract` — a lower type entity
  - if `Body` is a `Solid` and `index > 0`
    - create `Sheet` from `+index`'th `Face`
  - elseif `Body` is a `Solid` and `index < 0`
    - create `Wire` from `-index`'th `Edge`
  - elseif `Body` is `Solid` and `index = 0`
    - create `Sheet` from outer `Shell` of `Body`
  - elseif `Body` is `Sheet` and `index > 0`
    - create `Sheet` from `+index`'th `Face`
  - elseif `Body` is `Sheet` and `index < 0`
    - create `Wire` from `-index`'th `Edge`
  - elseif `Body` is `Sheet` and `index = 0`
    - create `Wire` from outer `Loop` (currently not implemented)



# Transformations

- Pops one Body off the stack
- Pushes the transformed Body 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)
  - `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`





# Applied Features (1)

- Pops one Body off the stack
- Pushes the modified Body onto the stack
- Supported applied features include:
  - `fillet` — round-over or fill-in Edges with a circular feature
    - by default (`edgeList=0`), this is applied to the Edges that were created in the previous Branch
    - for a Boolean, the “new” Edges that were created by the Boolean
    - for a primitive, can be applied to selected Edges by providing a semicolon-separated list of integers (in `edgeList`) with the following meaning:
      - `0; 0;` — add all Edges to the list
      - `+m; +n;` — add Edges between Faces *m* and *n* to the list
      - `-m; -n;` — remove Edges between Faces *m* and *n* to the list
      - `+m; 0;` — add all Edges adjacent to Face *m* to the list
      - `-m; 0;` — remove all Edges adjacent to Face *m* to the list



## Applied Features (2)

- Other supported applied features include:
  - `chamfer` — “knock-off” off or fill-in Edges with a diagonal feature
    - same rules as `fillet`
  - `hollow` — create a Body made by “hollowing-out” a Solid Body
    - if `thick` is zero, a Solid Body is converted to a Sheet Body or a Sheet Body is converted to a Wire Body
    - if `faceList` is zero (the default), create an “offset” Body
    - if `faceList` is a semicolon-separated list, remove the indicated Face(s)
    - Note: `hollow` does not work currently on Windows



## Miscellaneous Branches (1)

- `set` — set the value of a local variable to the given expression
- `mark` — push a Mark onto the stack
- `dump` — write file that contains the Body on the top of the stack
  - if `remove` is not zero, the Body is popped off the stack
- `store` — remember the identity of the Body on the top of the stack
  - each storage location has a name and an optional index
  - depending on the value of `keep`, the Body on the top of the stack is either kept (like a “copy”) or popped off the stack (like a “cut”)
  - this command is typically used in conjunction with the `restore` primitive



## Miscellaneous Branches (2)

- `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

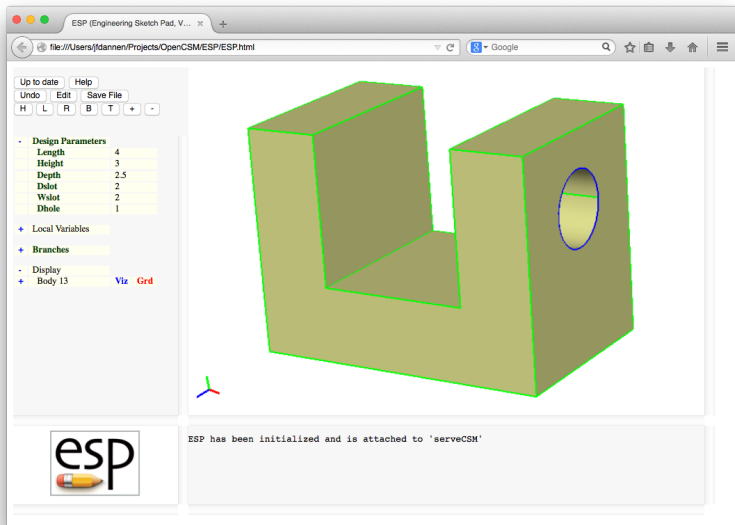


# Hands-on Exercises

- L-shaped bracket with hole
- bolt
- simple wing



# L-shaped Bracket with Hole (1)





## L-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	



## L-shaped Bracket with Hole (3)

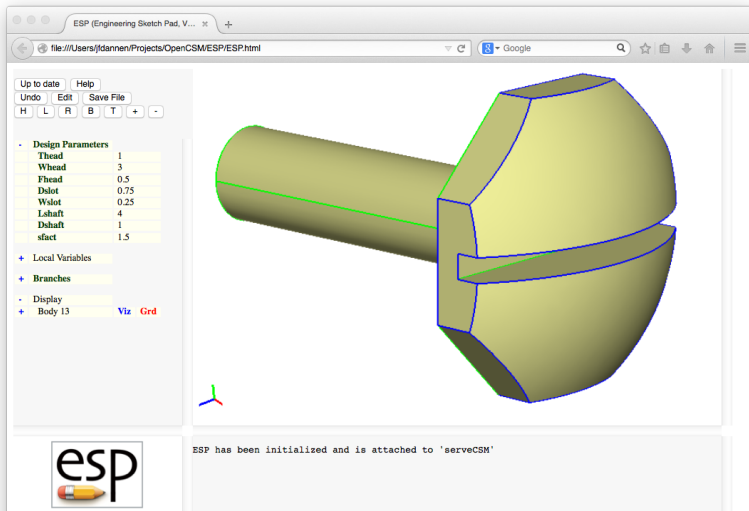
- What happens when you change a Design Parameter?
- What happens if you make Dhole large?
- Can you make the configuration without a Sketch?
- What happens if you add a fillet after the extrude?
- What happens if you add a fillet after subtracting the hole?





# Bolt (1)

Hint: If you get stuck, look back at previous slides



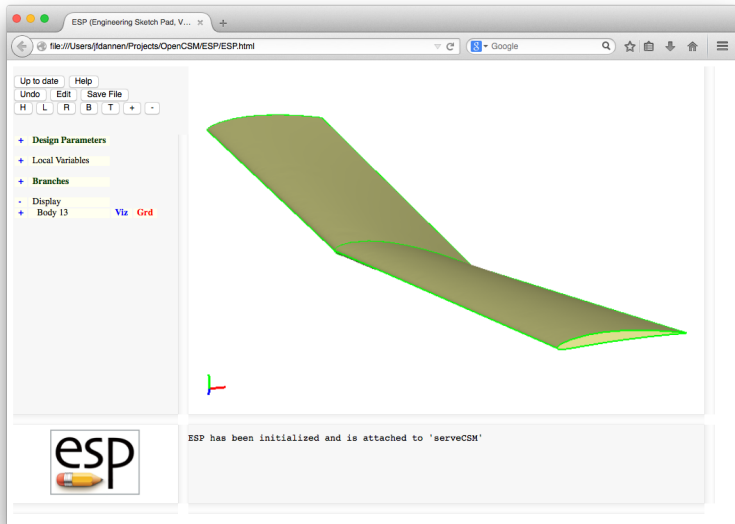


## Bolt (2)

Thead	axial thickness of the head	1.00
Whead	width of the head	3.00
Fhead	fraction of head that is flat	0.50
Dslot	depth of slot	0.75
Wslot	width of slot	0.25
Lshaft	length of shaft	4.00
Dshaft	diameter of shaft	1.00
sfact	overall scale factor	0.50



# 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?



# Muddy Cards

- Did you like frequent stops to see “what happens”?
- What about solids is particularly confusing?
- Do you think that you have the knowledge to build a vehicle with a wing and fuselage?