Engineering Sketch Pad (ESP)

Training Session 1
ESP Overview & Getting Started

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

Bob Haimes
haimes@mit.edu
Massachusetts Institute of Technology

updated for v1.18

Overview

- ESP Overview
  - Background and Objectives
  - ESP Architecture
  - Distinguishing Features
- Starting ESP
- User Interface
  - Screen Layout
  - Image Manipulation
  - View Manipulation
- Getting Info
- StepThru Mode
- Journals & Exporting

Background (1)

- Over the past 40 years, there have been an increasingly-complex (complicated) series of "CAD" systems to support the geometry needs of the manufacturers of mechanical devices
  - CAD = "computer aided drafting"
  - CAD = "computer-aided drawing"
  - CAD = "computer-aided design"
  - CAD = "computer-aided development"
- "CAD" has sometimes been erroneously equated with geometry

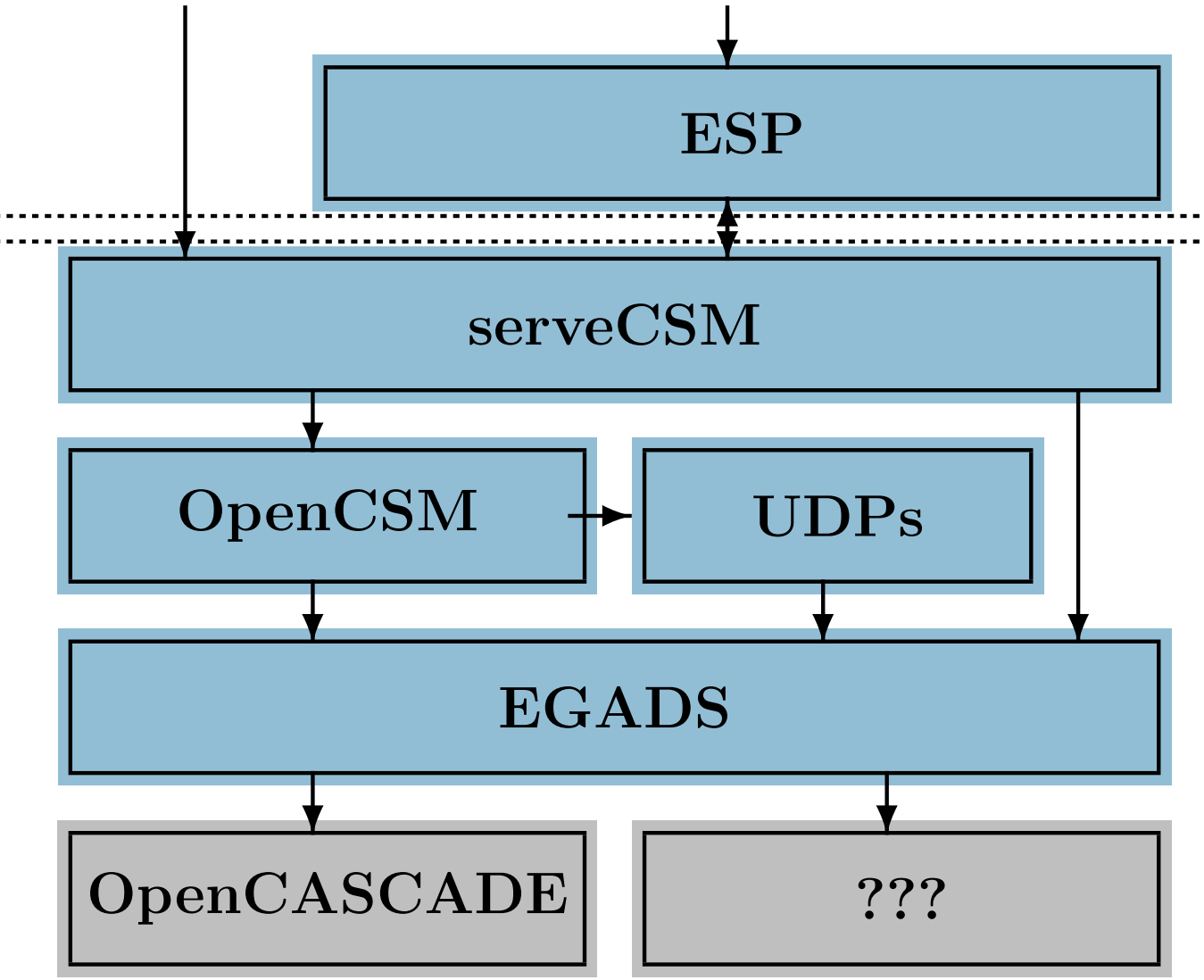
Background (2)

- These systems are built around the notion that the developer of a geometric model should construct the model to be consistent with the manufacturing process (**mCAD**)
- The analytical designer of a system wants to think about the function and performance of the device being generated, often leading to the generation of a separate **aCAD** model
- The modeling techniques supported by **aCAD** and **mCAD** are often so dissimilar that model transfer between them is done by limited translators or by "starting over"
- This one-way path from **aCAD** to **mCAD** leads to a "broken process"

Objective

- ESP is:
  - a geometry creation and manipulation system designed specifically to support the analysis and design of aerospace vehicles
  - can be run stand-alone for the development of models
  - can be embedded into other analysis and design systems to support their geometry needs
- ESP is not:
  - a full-featured computer-aided design (CAD) system designed specifically to support the mechanical design and manufacturing of any complex system
  - a system to be used for creating “drawings”

ESP Architecture



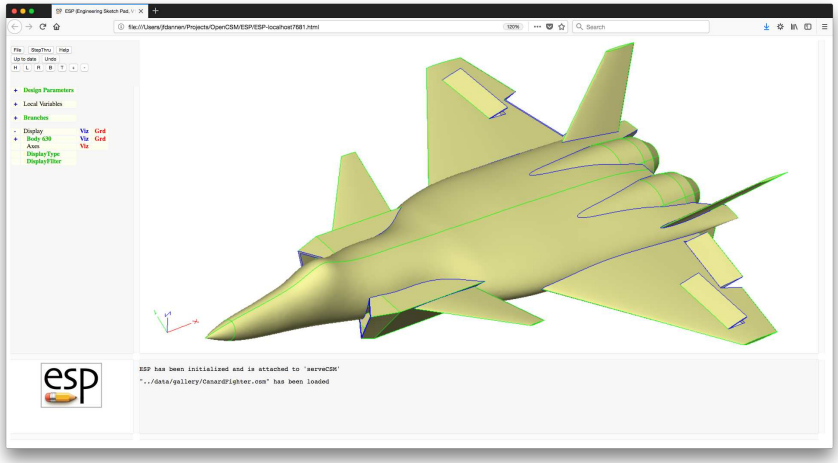
user interactions

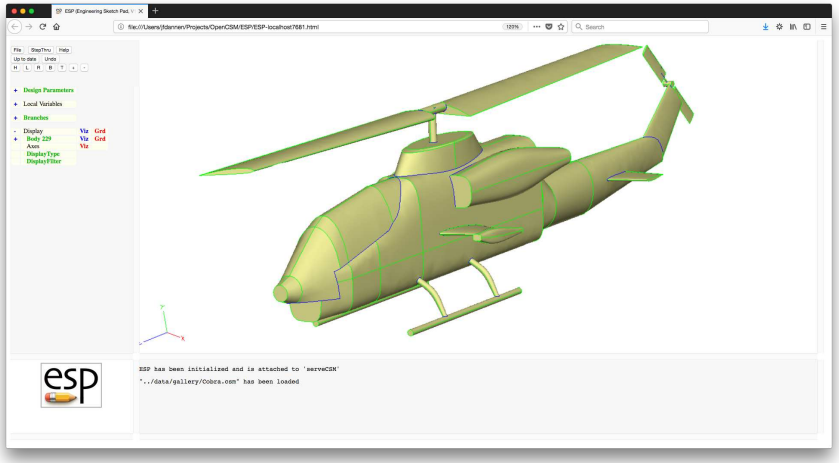
system driver
create graphic objects

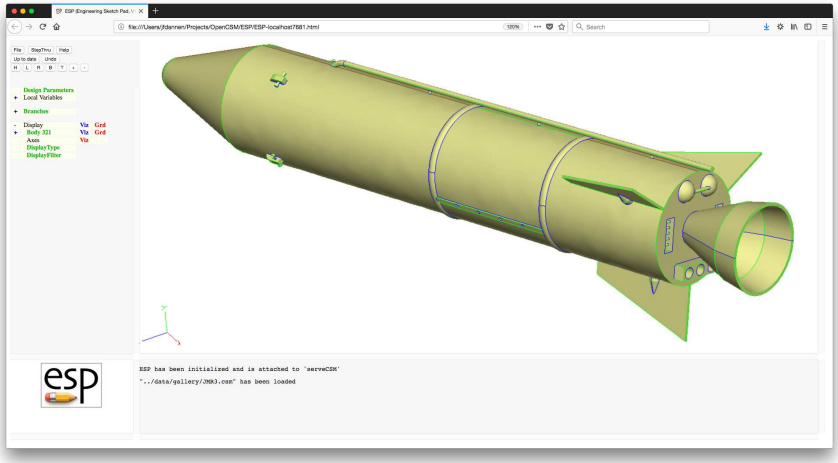
design parameters
feature tree
user-defined primitives
sensitivities
top-down & bottom-up
persistent attribution
watertight tessellation

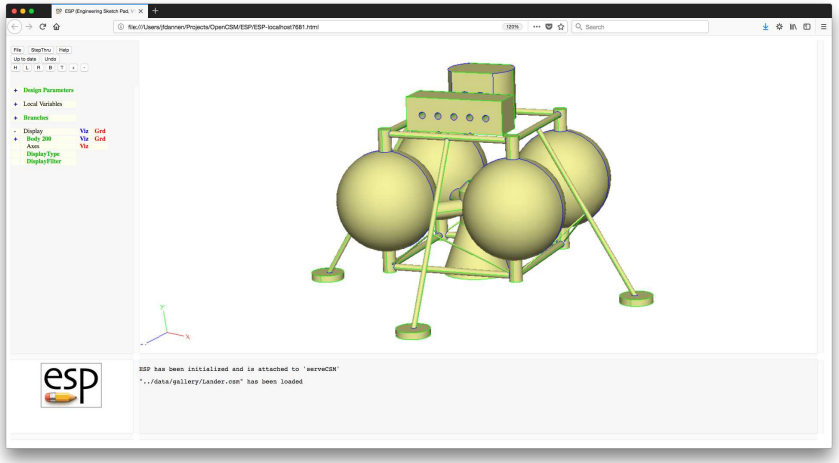
geometry primitives
boundary representations

Gallery of ESP Configurations









Distinguishing Features — Solid Modeller

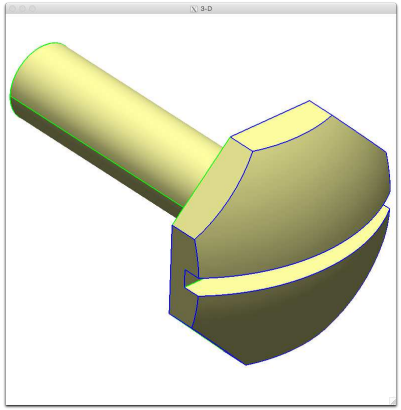
- Construction process guarantees that models are realizable solids
  - watertight representation needed for grid generators
  - sheets and wires are supported when needed
- Parametric models are defined in terms of:
  - Feature Tree
    - "recipe" for how to construct the configuration
  - Design Parameters
    - "values" that describe any particular instance of the configuration

Distinguishing Features — Feature-based

- Configurations start with the generation of primitives
  - standard primitives: box, sphere, cone, cylinder, torus
  - grown primitives (from sketches): extrude, rule, blend, revolve, sweep, loft
  - user-defined primitives (UDPs)
- Bodys can be modified
  - transformations: translate, rotate, scale, mirror
  - applications: fillet, chamfer, hollow
- Bodys can be combined
  - Booleans: intersect, subtract, union
  - other: join, connect, extract, combine

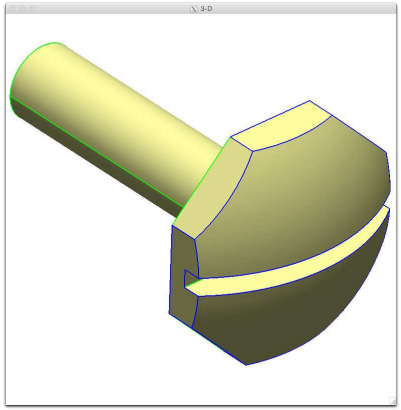
Construction Process (1)

```
      # 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

...
```

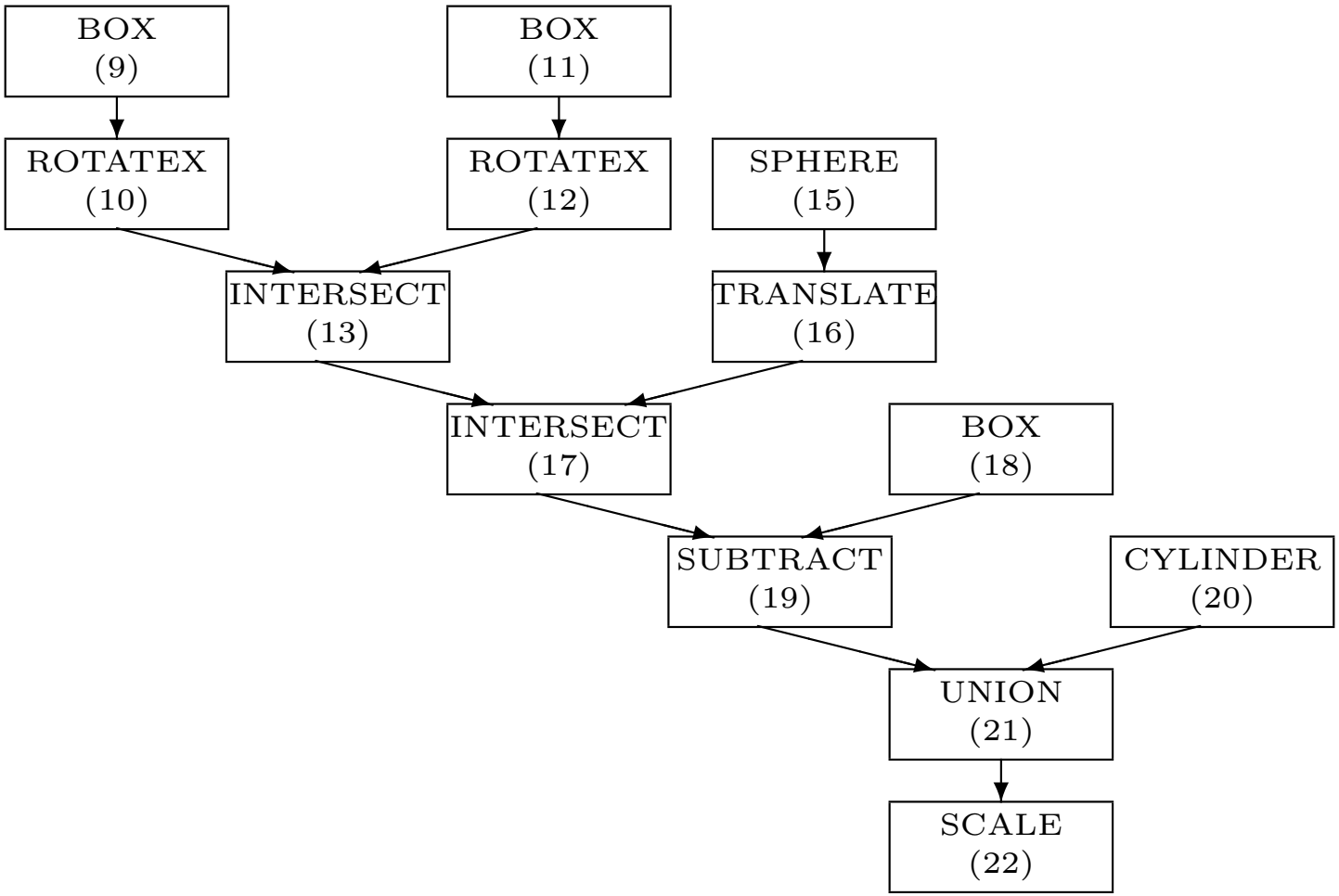


Construction Process (2)

```
...

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
```



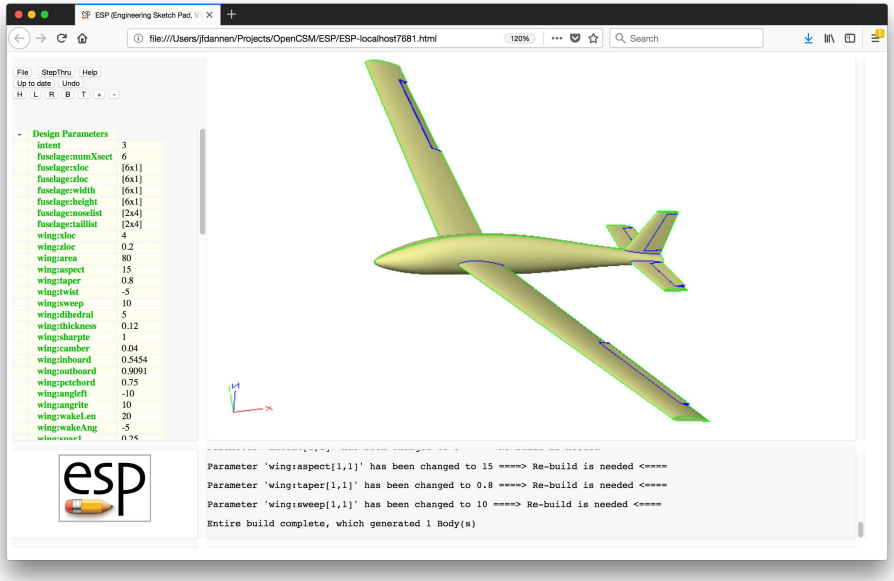
Review of Construction Process (3)



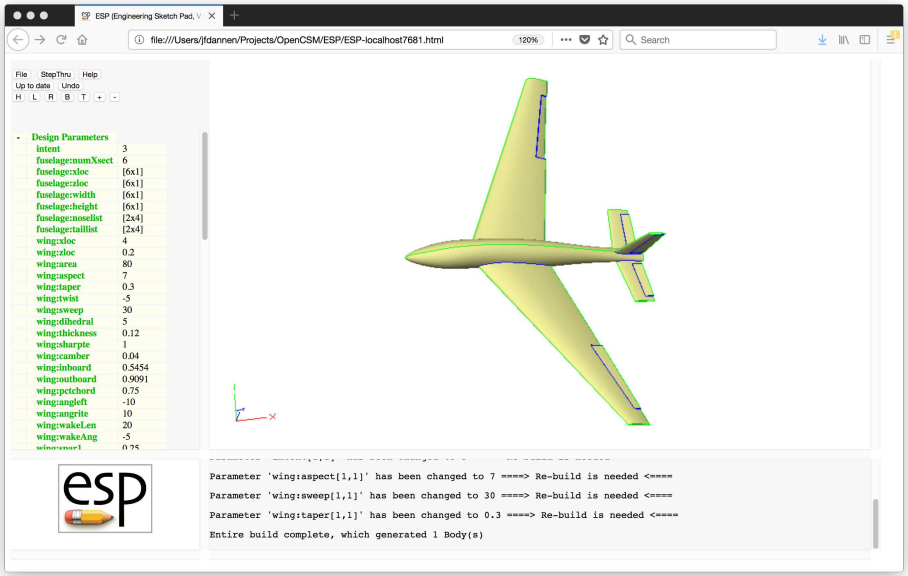
Distinguishing Features — Parametric

- ESP models typically contain one or more Design Parameters
- Design Parameters can be single-valued, 1D vectors, or 2D arrays of numbers
- Each Design Parameter has a current value, upper- and lower-bounds, and a current "velocity" (which is used to define sensitivities)
- Design Parameters can be "set" and "get"
  - through ESP's tree window
  - externally via calls to the Application Programming Interface (API)
- Arguments of all operations can be written as "expressions" that reference Design Parameters

Parameter Changes for Glider



$$\text{aspect} = 15$$
$$\text{sweep} = 10$$
$$\text{taper} = 0.8$$

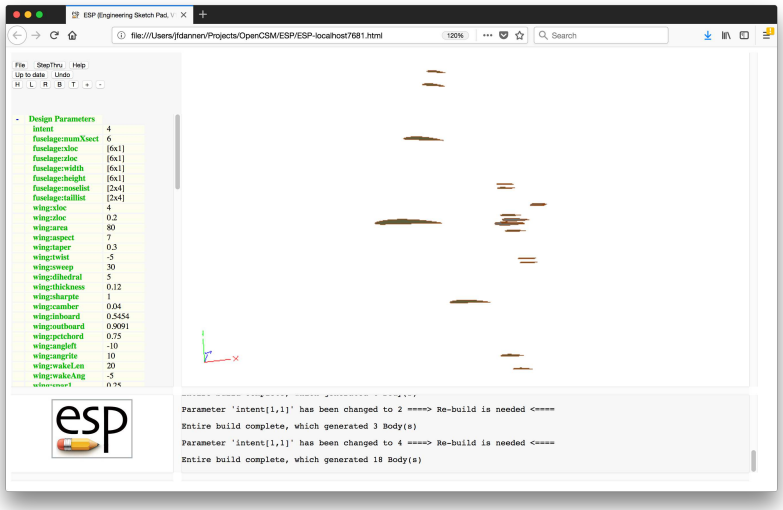


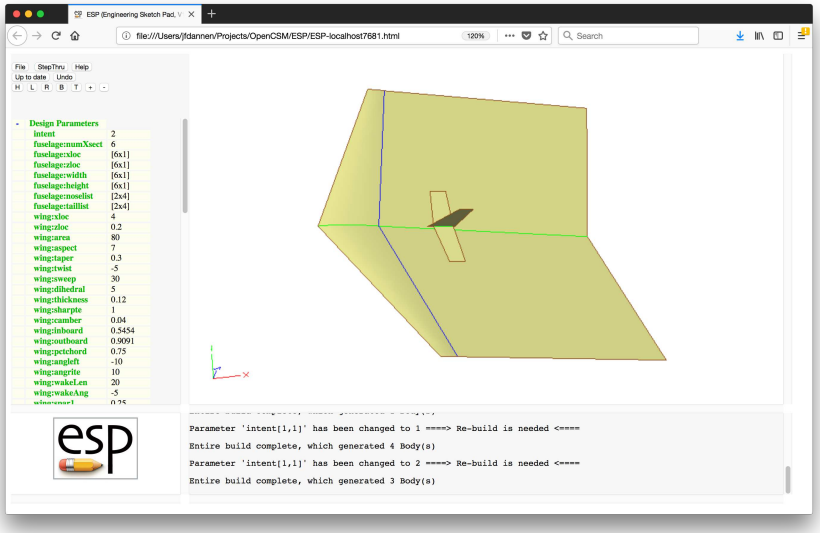
$$\text{aspect} = 7$$
$$\text{sweep} = 30$$
$$\text{taper} = 0.3$$

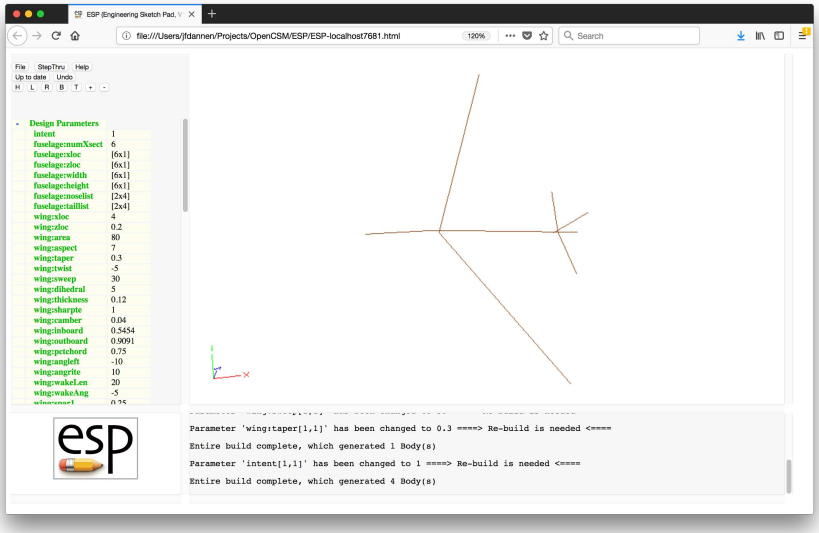
Distinguishing Features — Associative

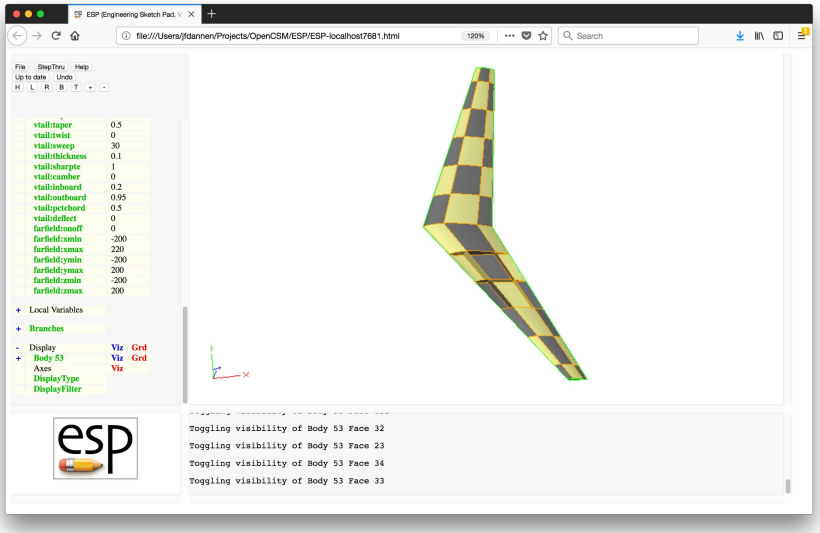
- ESP maintains a set of global and local attributes on a configuration that are persistent through rebuilds
- Supports the generation of multi-fidelity models
  - attributes can be used to associate conceptually-similar parts in the various models
- Supports the generation of multi-disciplinary models
  - attributes can be used to associate surface groups which share common loads and displacements
- Supports the "marking" of Faces and Edges with attributes such as nominal grid spacings, material properties, ...

Multiple Models for Glider





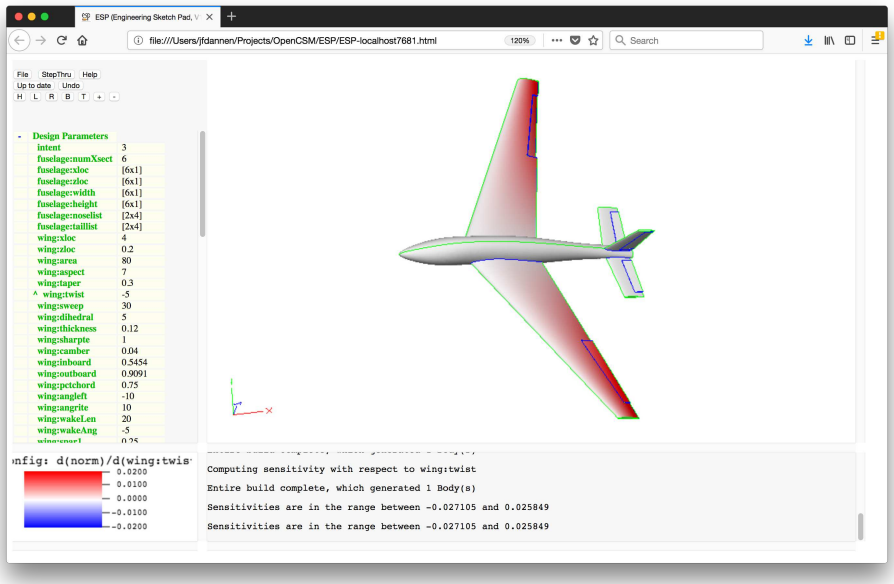




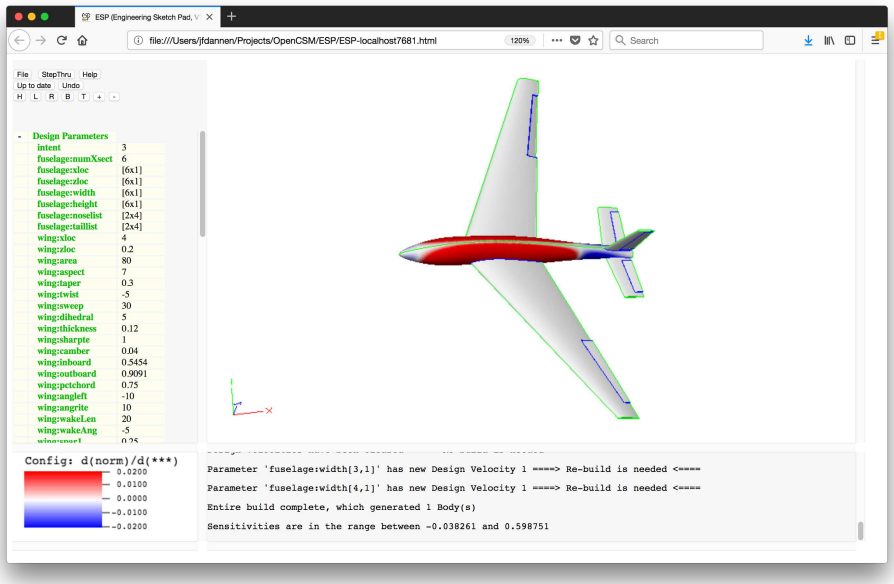
Distinguishing Features — Differentiated

- `ESP` allows a user to compute the sensitivity of any part of a configuration with respect to any Design Parameter
- Many of `OpenCSM`'s commands have been analytically "differentiated"
  - efficient, since there is no need to re-generate the configuration
  - accurate, since there is no truncation error associated with "differencing"
- Other commands (currently) require the use of finite-differenced sensitivities
  - robust, due to new mapping technique
  - less efficient, since it requires the generation of a "perturbed" configuration
  - less accurate, since one needs to carefully select a "perturbation step" that is a balance between truncation and round-off errors

Sensitivities for Glider



twist



fuselage width

Distinguishing Features — Extensible

- Users can add their own user-defined primitives (UDPs)
  - create a single primitive solid
  - are written in C, C++, or FORTRAN and are compiled
  - can be written either top-down or bottom-up
  - 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)
  - consume one or more Bodys from stack
  - are otherwise similar to UDPs
- Users can add their own user-defined components (UDCs)
  - can be thought of as "macros"
  - create zero or more Bodys
  - are written as `.csm`-type scripts

Distinguishing Features — Deployable

- ESP's back-end (server) runs on a wide variety of modern compute platforms
  - LINUX
  - OSX
  - Windows
- ESP's user-interface (client) runs in most modern web browsers
  - FireFox
  - Google Chrome
  - Safari
  - Note: Internet Explorer and Microsoft Edge are not supported at this time
- ESP can be distributed anywhere in the computer environment
  - open-source project (using the LGPL 2.1 license) that is distributed as source

Distinguishing Features — Embeddable

- Models are defined in `.csm` files
  - human readable ASCII
  - stack-like language that is consistent with Feature Tree traversal
  - contains looping via "patterns"
  - contains logical (if/then) constructs
  - contains error recovery via thrown/caught signals
- `OpenCSM` modeling system is defined by an Application Programming Interface (API) that allows it to be embedded into other applications
  - load a Master Model
  - interrogate and/or edit the Master Model
  - execute the Feature Tree and create BRep(s)
  - interrogate the BRep(s)
  - "set" and "get" sensitivities

Launching ESP (1)

- Double-clicking `runESP118` icon on desktop
  - Automatically starts server and brings up browser
  - User can select **File→Open** to use existing `.csm` file
  - Closing the browser automatically stops the server
  - No command-line options are used
- Double-clicking on `ESP118` icon on desktop
  - Brings up a terminal window in which all the `ESP` environment variables are set
  - Allows user to launch `serveCSM` multiple times, with filenames and/or command-line options
  - Terminal window remains open until the user closes it

Launching ESP (2)

- If starting from terminal window:
  - Technique 1: start browser automatically:

```
setenv ESP_START "open -a /Applications/Firefox.app ...
                                ... $ESP_ROOT/ESP/ESP.html"
```

    or

```
export ESP_START="open -a /Applications/Firefox.app ...
                                ... $ESP_ROOT/ESP/ESP.html"
```

    or

```
set ESP_START="open -a /Applications/Firefox.app ...
                              ... $ESP_ROOT/ESP/ESP.html"
```

    and then

```
serveCSM $ESP_ROOT/data/tutorial1
```

  - Technique 2: start browser separately:

```
serveCSM $ESP_ROOT/data/tutorial1
```

    and then open a browser on `ESP.html`

ESP Command Line (1)

- To start `serveCSM`

  `serveCSM [filename[.csm]] [options...]`

  where `[options...]` include:

  - `filename` is the name of the `.csm` file that contains the Model
  - `-batch` runs the case but does not attach to a browser
  - `-help` or `-h` prints listing of acceptable options
  - `-jrnl jrnlname` can be used to replay a previous session
    - current session is stored in file `portXXXX.jrnl`
    - file must be renamed to be used for next session
  - `-skipBuild` to skip initial build
  - `-skipTess` to skip tessellation at end (and automatically select `-batch`)
  - `--version` or `-version` or `-v` to return version information
  - ...

ESP Command Line (2)

- Other `[options...]` include:
  - `-despmtrs despname` to update the Design Parameters from the `despname` file
  - `-dict dictname` loads Constant Parameters from the `dictname` file file
  - `-dumpEgads` to dump `EGADS` file in form `Body_XXXXXX.egads` after each Body is built
  - `-loadEgads` to load `Body_XXXXXX.egads` file if it exists in current directory
  - `-onormal` to plot in (nearly) orthonormal (not perspective)
  - `-outLevel n` selects the output level (1 is the default)
  - `-port portnum` selects the port for communication with the browser (7681 is the default)
  - `-printStack` to print the contents of the stack after every command is executed (useful for debugging)

ESP Command Line (3)

- Other `[options...]` include:
  - `-plot plotfile` to plot additional information or provide input for the `-histDist` or `-plugs` options
  - `-plotBDF filename` superimposes BDF information in Graphics Window
  - `-plotCP` to plot Bspline control points
  - `-sensTess` to produce configuration sensitivity (instead of tessellation sensitivity) output
  - `-histDist dist` to generate histograms of the points in the `plotfile` from the configuration. Points that are further than `dist` are added to a new `plotfile` called `bad.points`
  - `-plugs npass` to use the experimental Plugs program, in which the values of the Design Parameters are adjusted so as to minimize the distance of the points in `plotfile` from the configuration.

ESP Command Line (4)

- Still other (less frequently used) `[options...]` include:
  - `-verify` to execute `ASSERT` statements that contain `verify=1`
  - `-addVerify` creates verification files (for automatic regression testing)
  - `-egg eggname` uses an external grid generator

ESP Command Line (5)

- Other (for development) `[options...]` include:
  - `-checkMass` to compare the internally computed mass properties with those computed via the tessellation
  - `-checkPara` to check the parallelizability of the build
  - `-ptrb ptrbname` to generate information with which the sensitivities are debugged

ESP Screen Layout

- Graphics window
  - 3D image
  - 2D sketcher
  - forms
- Tree window
  - Design Parameters
  - Local Variables
  - Branches
  - Display
- Key window
  - color key
- Messages window

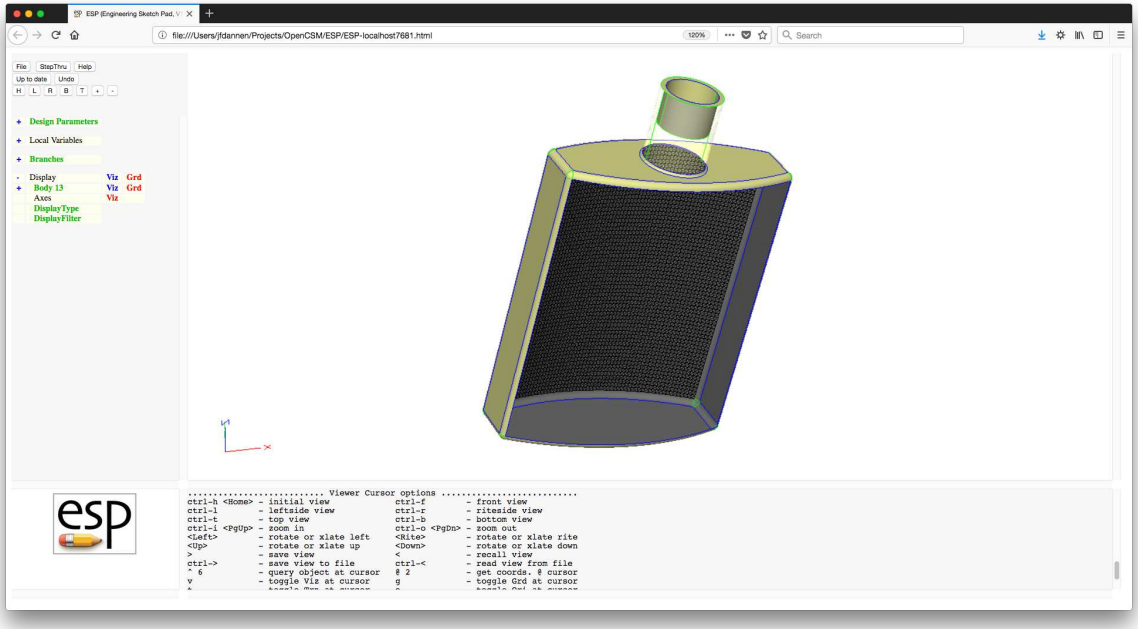


Image Manipulation via the Mouse

- Translation
  - press and drag any mouse button
- Rotation
  - hold down **Ctrl** and drag any mouse button
  - hold down **Alt** and drag any mouse button
- Zoom
  - hold down **Shift** and drag any mouse button
  - scrolling the middle mouse button also scrolls in/out
- Flying mode
  - press **!** in Graphics window to toggle mode
  - image continues moving image until mouse is released
- Note: the mouse mappings are defined in `ESP.js`

Image Manipulation via Key Presses (1)

"flying-mode" is off by default

Key-press	"flying-mode" off	"flying-mode" on
$\leftarrow$	rotate left 30°	translate left
$\rightarrow$	rotate right 30°	translate right
$\uparrow$	rotate up 30°	translate up
$\downarrow$	rotate down 30°	translate down
+	zoom in	zoom in
-	zoom out	zoom out
PgUp	zoom in	zoom in
PgDn	zoom out	zoom out
Home	home view	home view

Note: holding **Shift** reduces the increment

Image Manipulation by Key Presses (2)

Key-press	orientation	note
Ctrl-h	home view	y vs x
Ctrl-f	front view	y vs x
Ctrl-l	left side view	y vs z
Ctrl-r	right side view	y vs $-z$
Ctrl-b	bottom view	z vs x
Ctrl-t	top view	$-z$ vs x
Ctrl-i	zoom in	
Ctrl-o	zoom out	

Note: some of these signals may be intercepted by your browser

Image Manipulation via Buttons

Button press	orientation	note
H	home view	y vs x
L	left side view	y vs z
R	right side view	y vs $-z$
B	bottom view	z vs x
T	top view	$-z$ vs x
+	zoom in	
-	zoom out	

Buttons are near top of Tree window

Saving Viewing Orientation

key press	action
>	save view (in memory)
<	restore view (from memory)
Ctrl->	save view (in a file)
.	save view (in a file)
Ctrl-<	restore view (from a file)
,	restore view (from a file)

Image Manipulation via the Tree Window

- In the Tree window, **Display** contains an entry for each Body
- If the **Body** is expanded (the + on the left is pressed), then entries appear for **Faces**, **Edges**, **Nodes**, and **Csystems**
- If the **Faces**, **Edges**, **Nodes**, or **Csystems** are expanded, the names of all entities in the "group" are listed
- **Viz** toggles the visibility of the associated Body(s), Face(s), Edge(s), Node(s), or Csystem(s)
- **Grd** toggles the visibility of the grid of the associated Body(s), Face(s), or Edge(s)
- **Trn** toggles the pseudo-transparency of the associated Face(s)
- **Ori** toggles the orientation vectors of the associated Edge(s)
- Toggling at a "group" level effects the setting of its children
- Pressing **Display** gives the user the option of turning on/off the display of all Nodes, Edges, or Faces in all Bodys

Image Inquiry

- Re-center the image at the current location and set a new "rotation center"
  - * or **8**
- Find the location of the cursor (in 3D space) and report it in the Messages window
  - @ or **2**
- Identify the object (Edge or Face) and list all its attributes in the Messages window
  - $\wedge$ or **6**
- List the key-press options in the Messages window
  - **?**
- Orientation of image in Graphics window
  - red axis in x-direction
  - green axis in y-direction
  - blue axis in z-direction

Image Manipulation

- Turn off the visibility of the Node, Edge, or Face at cursor
  - **v**
- Toggle the grid on the Edge or Face at cursor
  - **g**
- Toggle the transparency of the Face at cursor
  - **t**
- Toggle the orientation of the Edge at cursor
  - **o**

StepThru Mode

- Show step-by-step build process
  - **StepThru** button (near top of Tree Window)
- Next step in build process
  - **NextStep** button (near top of Tree Window) or **n** key in Graphics Window
- Previous step in build process
  - **p** key in Graphics Window
- First step in build process
  - **f** key in Graphics Window
- Last step in build process
  - **l** key in Graphics Window
- Exit StepThru mode
  - **CancelStepThru** at bottom of Display listing in Tree Window

Creating a Script (1)

Using the ESP Interface

- Method:
  - start ESP: `serveCSM`
  - add Design Parameter by pressing **DesignParameters**
  - add Branch by pressing **Branch**
- Advantages:
  - most similar to other CAD packages
  - can use interactive sketcher
- Disadvantages:
  - generally slow
  - cannot add comments, indentation, etc.
  - harder to debug

Creating a Script (2)

Using an External Text Editor

- Method:
  - use any text editor to create `myFile.csm`
  - run ESP: `serveCSM -loadEgads-dumpEgads myFile`
- Advantages;
  - can use any editor with which you are familiar
  - easy to add comments, spacing, indentation, ...
- Disadvantages:
  - do not get help in writing `.csm` file
  - cannot use interactive sketcher
  - requires many `ESP` restarts

Creating a Script (3)

Using the Integrated Code Editor

- Method:
  - start ESP: `serveCSM`
  - **File→Edit** and then **Save**
- Advantages:
  - context-sensitive editor with hints
  - easy to add comments, spacing, indentation, ...
- Disadvantages:
  - slightly different key mappings
  - cannot use interactive sketcher

Using the jrnl (1)

- Every time that you execute `ESP`, a new `.jrnl` file is generated (which overwrites any existing file)
  - default name if `port7681.jrnl` (unless you used the `-port` command line option)
- The `.jrnl` file remembers all the interactions that you had with the `ESP` interface (example on next page)
- Each user action is a separate line in the `.jrnl` file

Using the jrnl (2)

Example `port7681.jrnl`

```
setPmtr|H|1|1|3|
build|0|
clrVels|
setVel|D|1|1|1|
build|0|
```

Using the jrnl (3)

- To use a `.jrnl` file, follow these steps:
  - when `ESP` completes, rename the `.jrnl` file, with a command such as

    ```
    mv port7681.jrnl my.jrnl
    ```

    or

    ```
    ren port7681.jrnl my.jrnl
    ```

    (this is needed so that the `.jrnl` is not overwritten below)
  - edit the `.jrnl` file to remove the offending command (which is usually the last line)
  - restart `ESP` with the command

    ```
    serveCSM -jrnl my.jrnl my.csm
    ```

    (assuming that the name of your `.csm` file is `my.csm`)

Saving vs. Exporting (1)

- ESP has two ways of saving your work:
  - File→Edit→Save
    - Save an exact copy of information in the code editor
    - Remembers comments, indentation, line-splitting, spacing, etc.
    - Is preferred method of saving your work, unless you make changes in the ESP Tree Window (for example, add/edit/remove a Branch or change a Design Parameter)
  - File→Export FeatureTree
    - Makes an output file by reading the current feature tree
    - Forgets comments, indentation, line-splitting, spacing, etc.
    - Is only useful if you have made edits via the Tree Window

Saving vs. Exporting (2)

Original `.csm` file

```
# example program
# written by John Dannenhoffer

# define parameters for the box
DESPMTR   L   3.0   # length (ft)
DESPMTR   H   2.0   # height (ft)
DESPMTR   D   1.0   # depth  (ft)

# create the box (centered at the origin)
BOX      -L/2  -H/2  -D/2 \
          L     H     D

# put _name attributes on the Faces
PATBEG       iface  6
   SELECT    FACE   iface
   ATTRIBUTE _name  $face_+iface
PATEND

END
```

Saving vs. Exporting (3)

.csm file generated by Export FeatureTree

```
# example_out.csm written by ocsmSave (v1.18)

# Constant, Design, and Output Parameters:
despmtr   L       3.00000
despmtr   H       2.00000
despmtr   D       1.00000

# Global Attributes:

# Branches:
box       -L/2   -H/2   -D/2   L   H   D
patbeg    iface   6
   select    FACE   iface
attribute _name   $face_+iface
patend

end
```

$ESP_ROOT/doc/ESP_QuickReference.pdf

CSM Commands

Primitives

POINT	xloc yloc zloc
BOX	xbase ybase zbase dx dy dz
SPHERE	xcent ycent zcent radius
CYLINDER	xbeg ybeg zbeg xend yend zend radius
CONE	xvrtx yvrtx zvrtx xbase ybase zbase radius
TORUS	xcent ycent zcent dxaxis dyaxis dzaxis ... ... majorRad minorRad
IMPORT	$filename bodynumber=1
UDPRIM	$primtype $argName1 argValue1 ...argValue4 name → UDP/UDF /name → path($pwd)/name.udc $/name → path($csm)/name.udc $$/name → path($root)/udc/name.udc
RESTORE	$name index=0

Grown

EXTRUDE	dx dy dz
RULE	reorder=0
BLEND	begList=0 endList=0 reorder=0 oneFace=0
REVOLVE	xorig yorig zorig dxaxis dyaxis dzaxis angDeg
SWEEP	
LOFT*	smooth

Applied

FILLET	radius edgeList=0 listStyle=0
CHAMFER	radius edgeList=0 listStyle=0
HOLLOW	thick=0 entList=0 listStyle=0

Booleans

INTERSECT	$order=none index=1 maxtol=0
SUBTRACT	$order=none index=1 maxtol=0
UNION	toMark=0 trimList=0 maxtol=0
JOIN	toler=0 toMark=0
CONNECT	faceList1 faceList2 edgeList1=0 edgeList2=0
EXTRACT	entList
COMBINE	toler=0

Transforms

TRANSLATE	dx dy dz
ROTATEX	angDeg yaxis zaxis
ROTATEY	angDeg zaxis xaxis
ROTATEZ	angDeg xaxis yaxis
SCALE	fact xcent=0 ycent=0 zcent=0
MIRROR	nx ny nz dist=0
APPLYCSYS	$csysName ibody=0
REORDER	ishift iflip=0

Sketch

SKBEG	x y z relative=0
SKVAR	$type vallist
SKCON	$type index1 index2=-1 $value=0
LINSEG	x y z
CIRARC	xon yon zon xend yend zend
ARC	xend yend zend dist $plane=xy
SPLINE	x y z
SSLOPE	dx dy dz
BEZIER	x y z
SKEND	wireonly=0

Solver

SOLBEG	$varList
SOLCON	$expr
SOLEND	

Stack

MARK	
STORE	$name index=0 keep=0
GROUP	nbody=0

Logic

IFTHEN	val1 $op1 val2 $op2=and val3 $op3 val4
ELSEIF	val1 $op1 val2 $op2=and val3 $op3 val4
ELSE	
ENDIF	

Looping

PATBEG	$pmtrName ncopy
PATBREAK	expr
PATEND	

Error handling

CATBEG	sigCode
CATEND	
THROW	sigCode

Declarations

DIMENSION	$pmtrName nrow ncol despmtr=0
CFGPMTR	$pmtrName values
DESPMTR	$pmtrName values
CONPMTR	$pmtrName value
OUTPMTR	$pmtrName
LBOUND	$pmtrName bounds
UBOUND	$pmtrName bounds

Attribution

ATTRIBUTE	$attrName attrValue
CSYSTEM	$csysName csysList
GETATTR	$pmtrName attrID global=0

User-defined components

INTERFACE	$argName $argType default=0
END	

Miscellaneous

SET	$pmtrName exprs
UDPARG	$primtype $argName1 argValue1 ...
SELECT	$type arg1 ...
ASSERT	arg1 arg2 toler=0 verify=0
DUMP	$filename remove=0 toMark=0
EVALUATE	$type arg1 ...
NAME	$branchName
PROJECT	x y z dx dy dz useEdges=0

User-defined Primitives/Functions

bezier	$filename debug imax jmax cp[]
biconvex	thick camber
box	dx dy dz rad @area @volume
csm	$filename $pmtrname pmtrvalue @volume
createBEM	$filename space imin imax nocrod
createPoly	$filename hole[]
droop	xle thetale xye thetate
editAttr	$attrname $input $output overwrite $filename verbose @nchange
ellipse	rx ry rz nedge thbeg
fitcurve	$filename ncp ordered periodic xform[] xyz[] @npnt @rms
flend	fraca fracb toler plot
freeform	$filename imax jmax kmax xyz[]
ganged	$op toler
guide	nxsect origin axis
hex	corners[] uknots[] vknots[] wknots[] @area @volume
import	$filename bodynumber @numbodies
kulfan	class[] ztail[] aupper[] alower[]
naca	series thickness camber maxloc offset sharpte
naca456	thkcode toc xmaxt leindex camcode cmax xmaxc cl a
nurbbody	$filename
parsec	yte poly[] param[] meanline
pod	length fineness @volume
poly	points[]
printBbox	
printBrep	

(continued on other side)

(UDPs/UDFs — continued from other side)

radwaf	ysize zsize nspoke xframe[]
sew	$filename toler bodynum
stag	rad1 beta1 gama1 rad2 beta2 gama2 ... ... alfa xfrnt xrear
stiffener	beg[] end[] depth angle
supell	rx rx_w rx_e ry ry_s ry_n n n_w n_e ... ... n_s n_n n_sw n_se n_nw n_ne offset nquad
waffle	depth segments[] $filename progress

Built-in Functions

General

```
pi(x)
min(x,y)
max(x,y)
sqrt(x)
abs(x)
int(x)
nint(x)
ceil(x)
floor(x)
mod(a,b)
sign(test)
exp(x)
log(x)
```

Trigonometric

```
log10(x)
sin(x)
sind(x)
asin(x)
asind(x)
cos(x)
cosd(x)
acos(x)
acosd(x)
tan(x)
tand(x)
atan(x)
atand(x)
atan2(y,x)
atan2d(y,x)
hypot(x,y)
hypot3(x,y,z)
```

Sketch utilities

```
incline(xa,ya,dab,xb,yb)
Xcent(xa,ya,dab,xb,yb)
Ycent(xa,ya,dab,xb,yb)
Xmidl(xa,ya,dab,xb,yb)
Ymidl(xa,ya,dab,xb,yb)
seglen(xa,ya,dab,xb,yb)
radius(xa,ya,dab,xb,yb)
sweep(xa,ya,dab,xb,yb)
turnang(xa,ya,dab,xb,yb,dbc,xc,yc)
dip(xa,ya,xb,yb,rad)
smallang(x)
```

Conversions

```
val2str(num,digits)
str2val(string)
findstr(str1,str2)
slice(str,ibeg,iend)
path($pwd) or path($csm) or path($root) or path($file)
```

Logic

```
ifzero(test,ifTrue,ifFalse)
ifpos(test,ifTrue,ifFalse)
ifneg(test,ifTrue,ifFalse)
ifmatch(str,pat,ifTrue,ifFalse)
ifnan(test,ifTrue,ifFalse)
```

Dot-suffixes

x.nrow	number of rows in x or 0 if a string
x.ncol	number of columns in x or 0 if a string
x.size	number of elements in x (=x.nrow*x.ncol) or len of str x
x.sum	sum of elements in x
x.norm	L2-norm (RMS) of elements in x
x.min	minimum value in x
x.max	maximum value in x

Character Set

#	hash	introduces comment
"	quotes	ignore spaces until following "
\	backslash	ignore this and following characters and concatenate next line
<space>	space	separates arguments in .csm file (except between " and ")
0-9		digits used in numbers, names, and strings
A-Z a-z		letters used in names and strings
_ : @		characters used in names and strings
? % =		characters used in strings
.	period	decimal separator (used in numbers), introduces dot-suffixes (in names)
,	comma	separates function arguments and row/column in subscripts
;	semicolon	multi-value item separator
( )	parentheses	groups expressions and function arguments
[ ]	brackets	specifies subscripts in form [row,column] or [index]
{ } < >		characters used in strings
+ - * / ^		arithmetic operators
$	dollar	as first character, introduces a string that is terminated by end-of-line or un-escaped plus, comma, or open-bracket
@	at-sign	as first character, introduces @-parameters
'	apostrophe	used to escape comma, plus, or open-bracket within strings
!	exclamation	if first character of implicit string, ignore $! and treat as an expression
|	bar	cannot be used (reserved for OpenCSM internals)
~	tilde	cannot be used (reserved for OpenCSM internals)
&	ampersand	cannot be used (reserved for OpenCSM internals)

ESP User Interface

Keypress (if not caught by browser)

ctrl-h -or- <Home>	initial view (or **H** button)
ctrl-f	front view
ctrl-l	leftside view (or **L** button)
ctrl-r	riteside view (or **R** button)
ctrl-t	top view (or **T** button)
ctrl-b	bottom view (or **B** button)
ctrl-i -or- <PgUp>	zoom in (or + button)
ctrl-o -or- <PgDn>	zoom out (or - button)
<Left>	rotate or xlate (in flying mode) left
<Rite>	rotate or xlate (in flying mode) rite
<Up>	rotate or xlate (in flying mode) up
<Down>	rotate or xlate (in flying mode) down
>	save view
<	recall view
ctrl-> -or- .	save view to file
ctrl-< -or- ,	read view from file
^ -or- 6	query object at cursor
@ -or- 2	get coords. @ cursor
v	toggle Viz (visability) at cursor
g	toggle Grd (grid) at cursor
t	toggle Trn (transparency) at cursor
o	toggle Ori (orientation) at cursor
A	add Attribute at cursor
* -or- 8	center view @ cursor
!	toggle flying mode

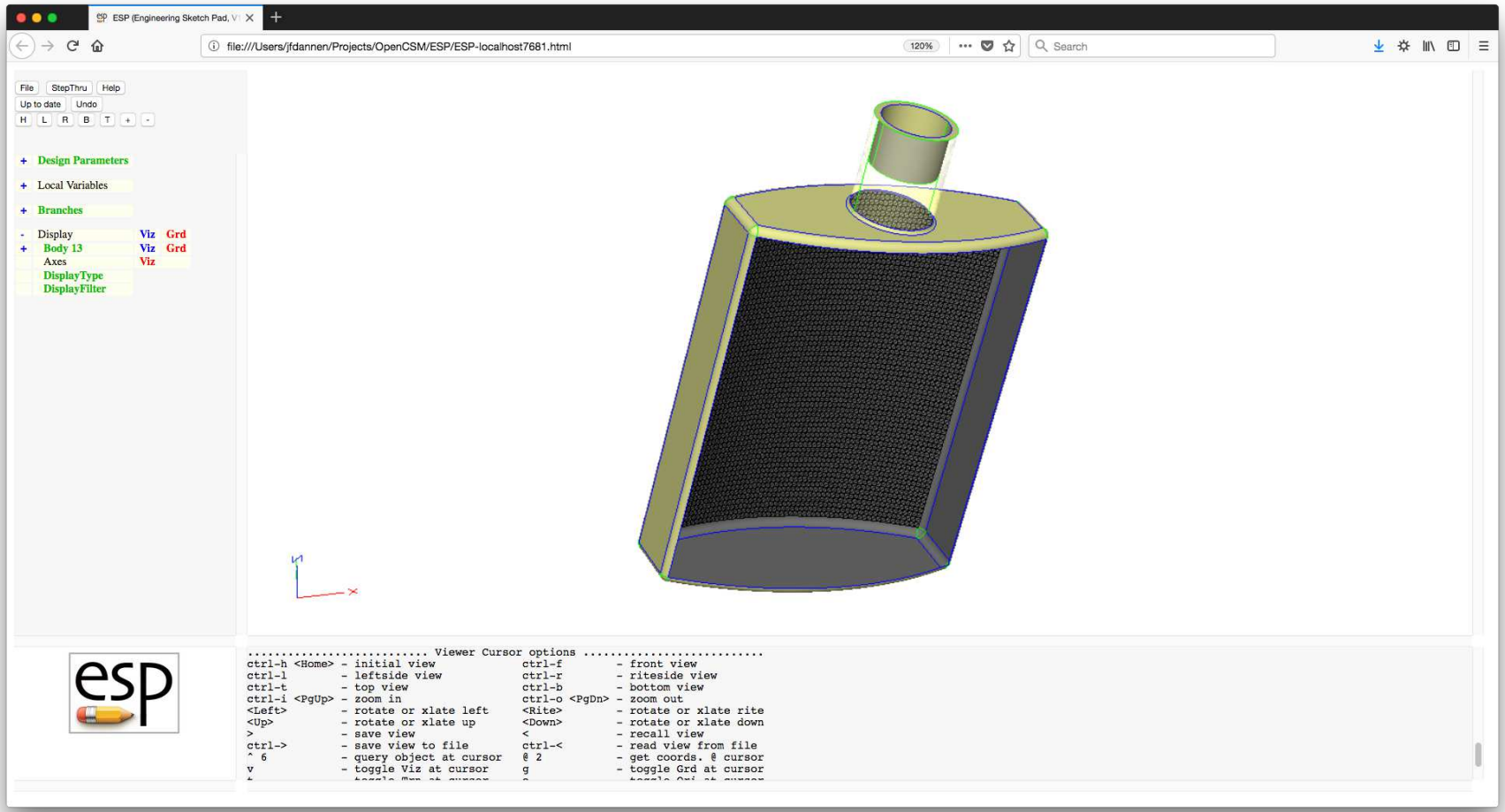
Recovering from an Error

- If the Message Window turns yellow
  - `OpenCSM` has detected an error
  - Double-clicking in the Message Window will automatically open the code editor to the appropriate line
- If the Message Window turns pink
  - `ESP` has lost its connection to `serveCSM` and the session must be restarted
  - Consider using the `-jrnl` option to get you (almost) back to the situation that caused the connection to be lost

Homework exercise

1. Start `serveCSM` using the file
   `$ESP_ROOT/training/ESP/data/session01/bottle.csm`
   or
   `../training/ESP/data/session01/bottle.csm`
2. Explore the various image manipulation tools
3. See if you can get the image on the next page
4. Use StepThru to see how the bottle was created

`bottle` After Image Manipulations



Muddy Cards

- Opportunity to provide immediate "feedback"
- Any questions about presentation material, critique of sample problems, ...
- Mail questions to `jfdannen@syr.edu`
- Questions will be answered at next session