Engineering Sketch Pad (ESP)

Training Session 1.3
ESP Overview & Getting Started

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

Bob Haimes
haimes@mit.edu

Marshall Galbraith
galbramc@mit.edu

Massachusetts Institute of Technology

Overview

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

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

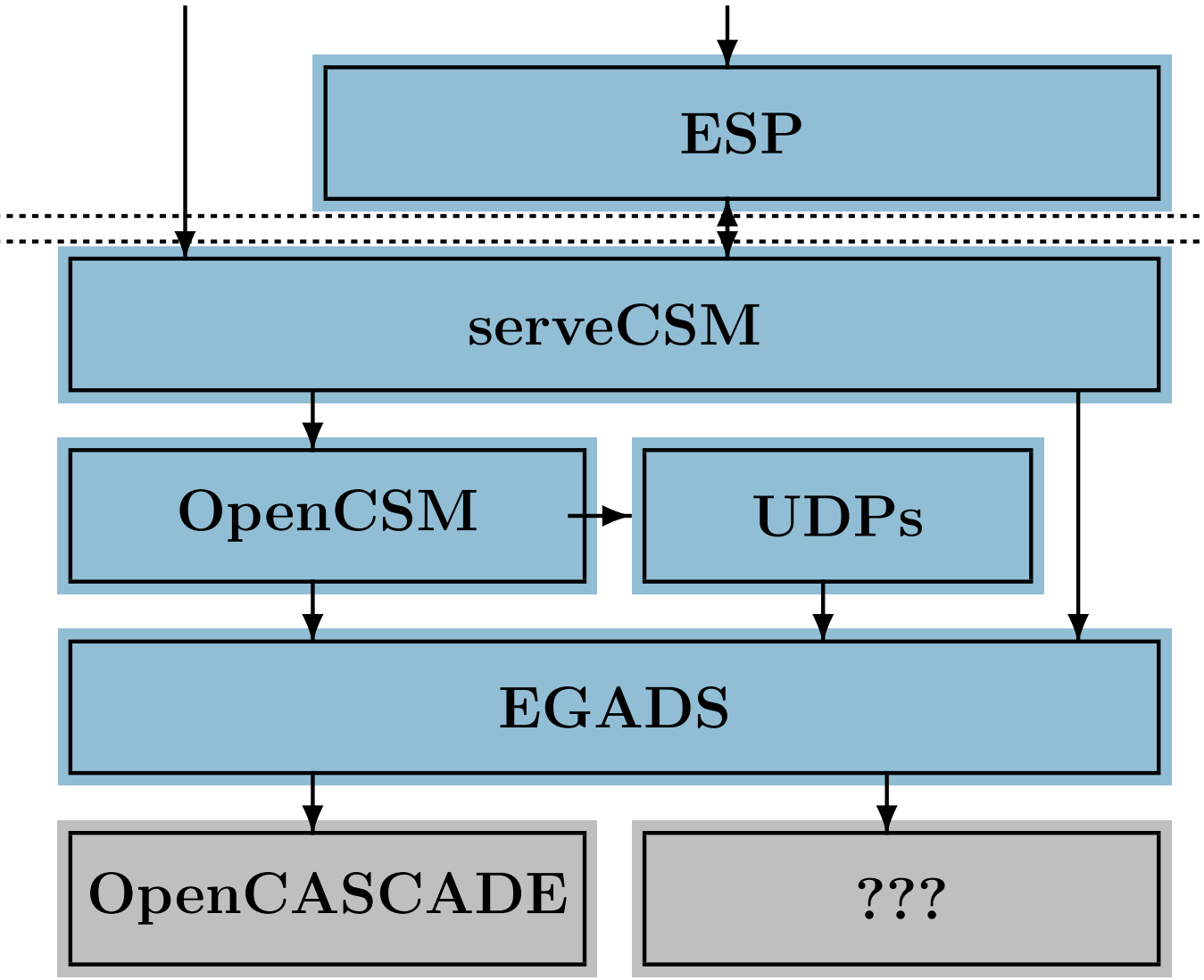
Background (2)

- 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



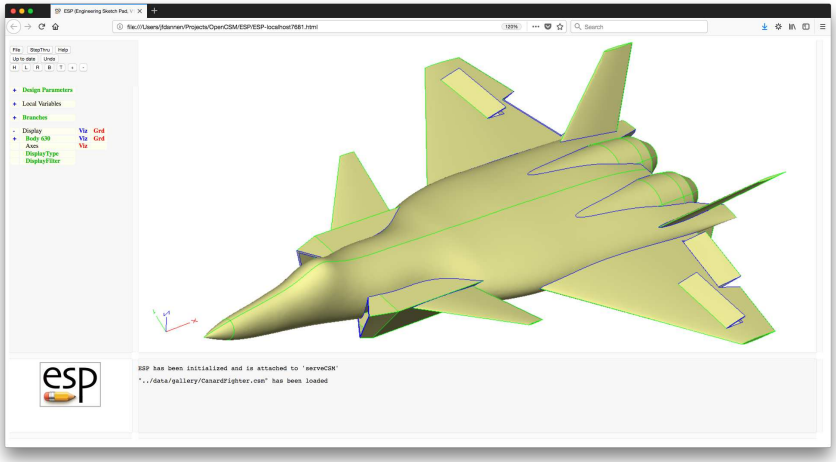
user interactions

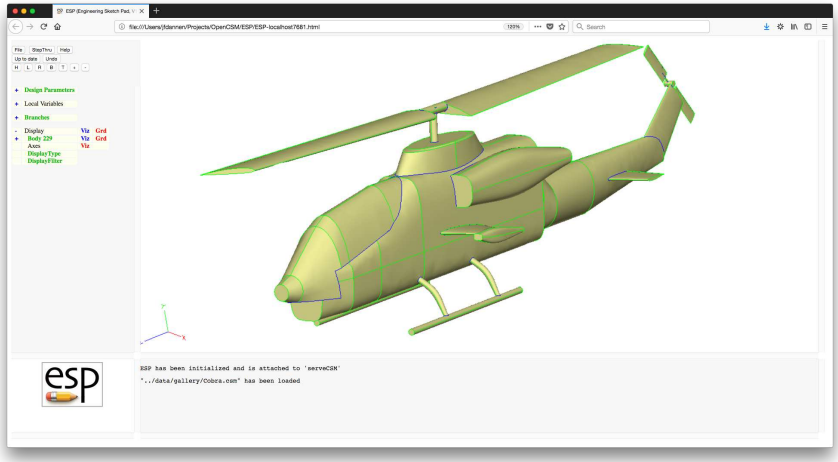
system driver
create graphic objects

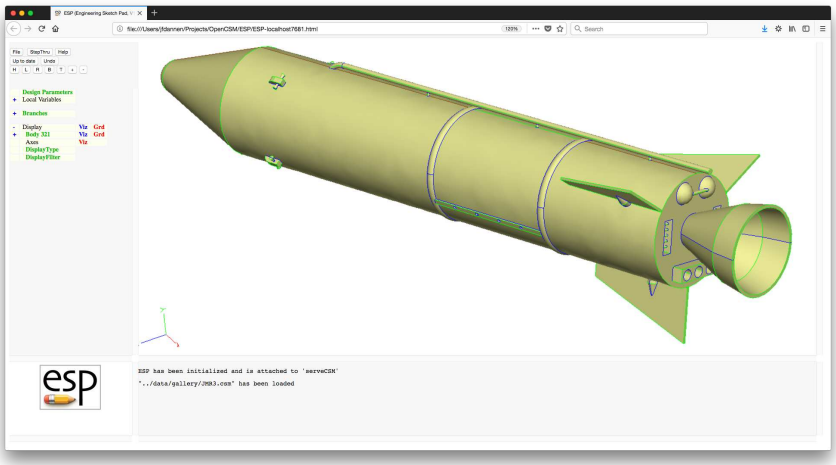
design parameters
feature tree
user-defined primitives
sensitivities
top-down & bottom-up
persistent attribution
watertight tessellation

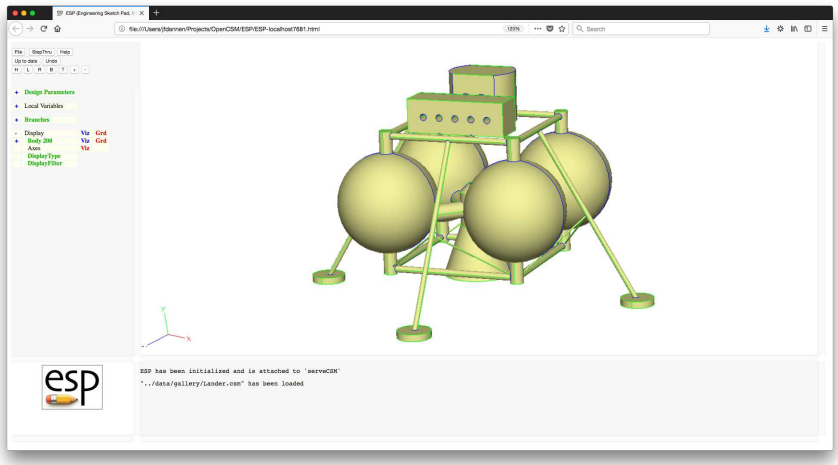
geometry primitives
boundary representations

Gallery of ESP Configurations









Distinguishing Features — Solid Modeller

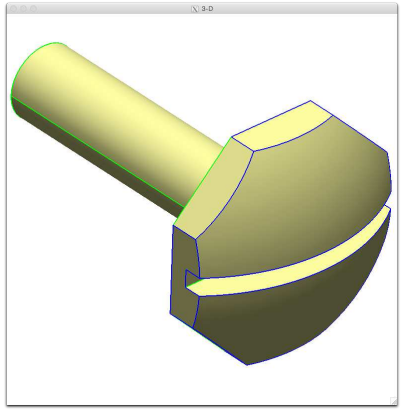
- Construction process guarantees that models are realizable solids
  - watertight representation needed for grid generators
  - sheets and wires are supported when needed
- Parametric models are defined in terms of:
  - Feature Tree
    - "recipe" for how to construct the configuration
  - Design Parameters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

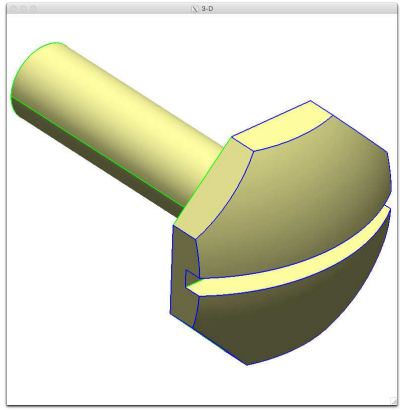
Construction Process (1)

```
      # 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

...
```

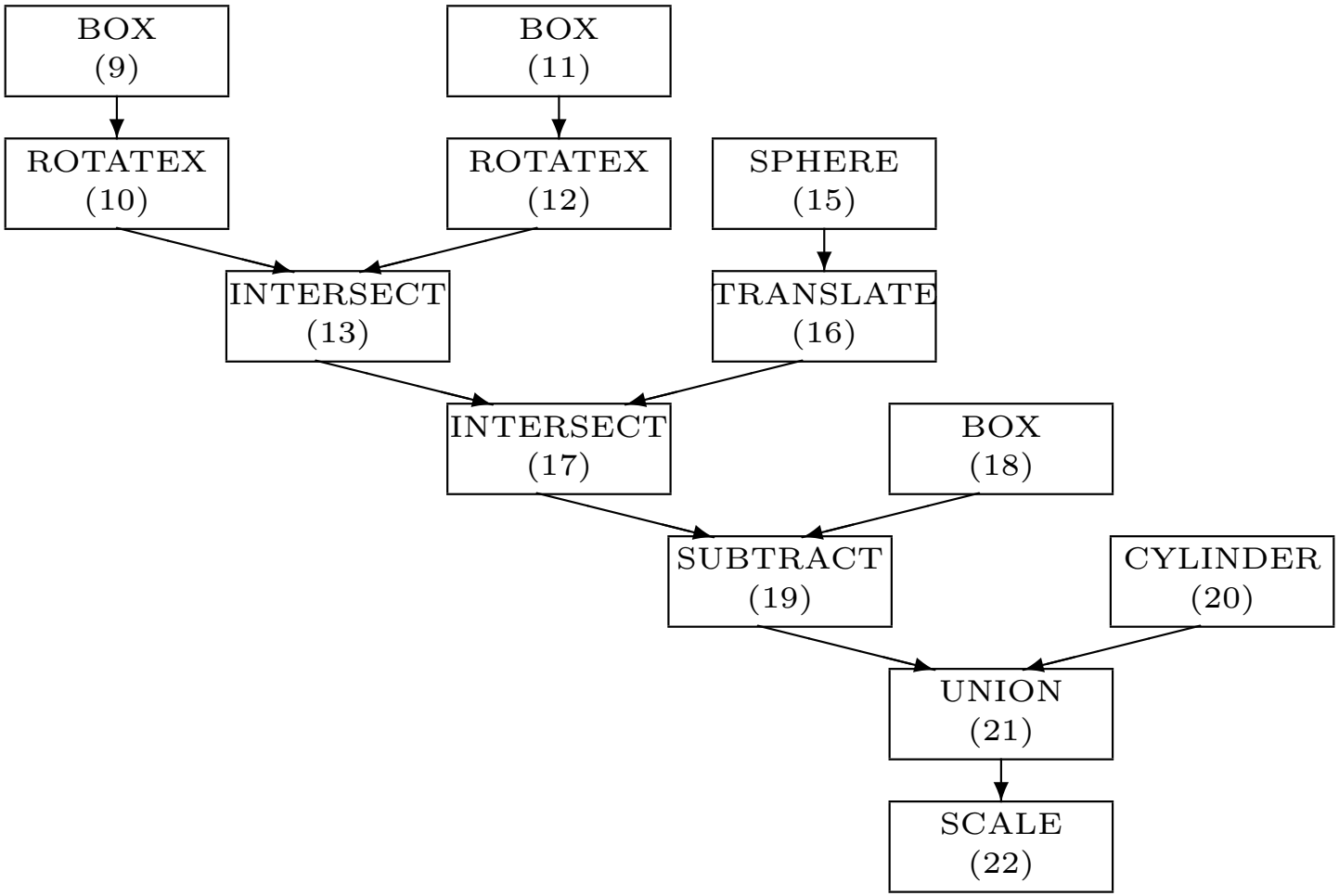


Construction Process (2)

```
...

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
```



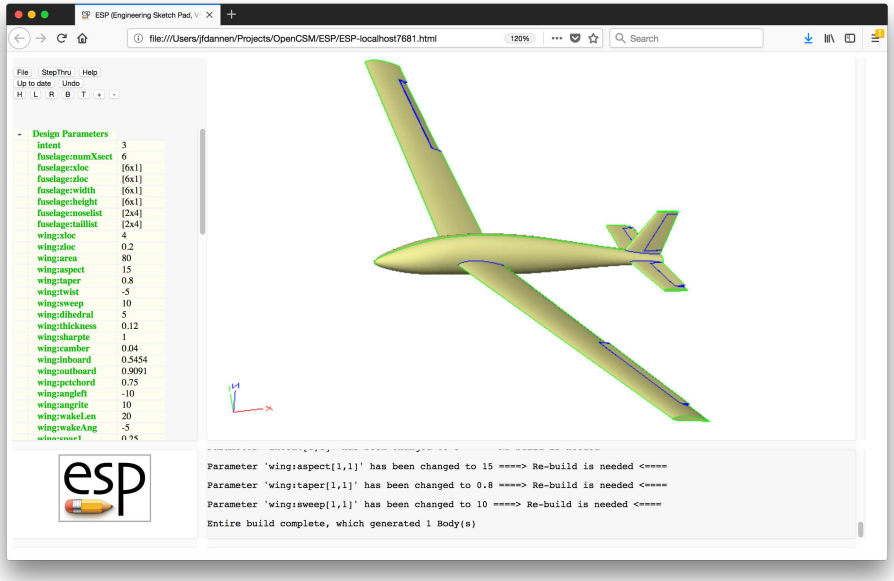
Review of Construction Process (3)



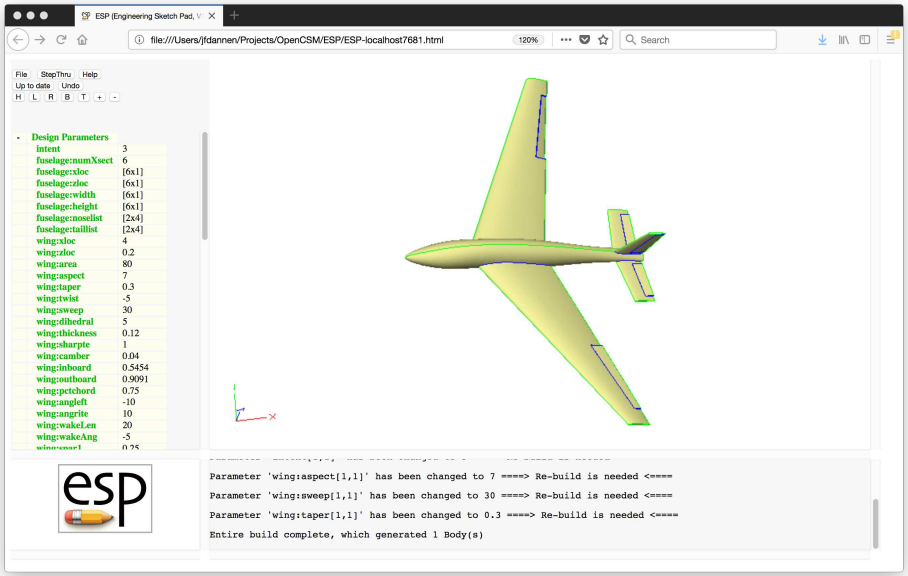
Distinguishing Features — Parametric

- 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



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

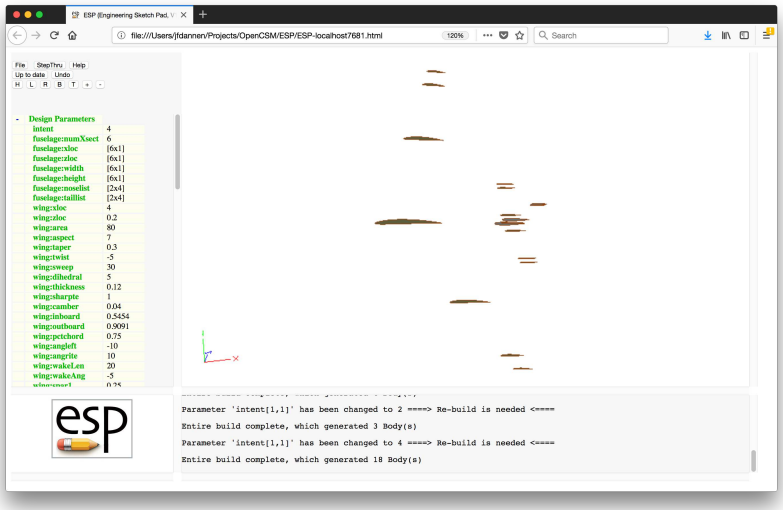


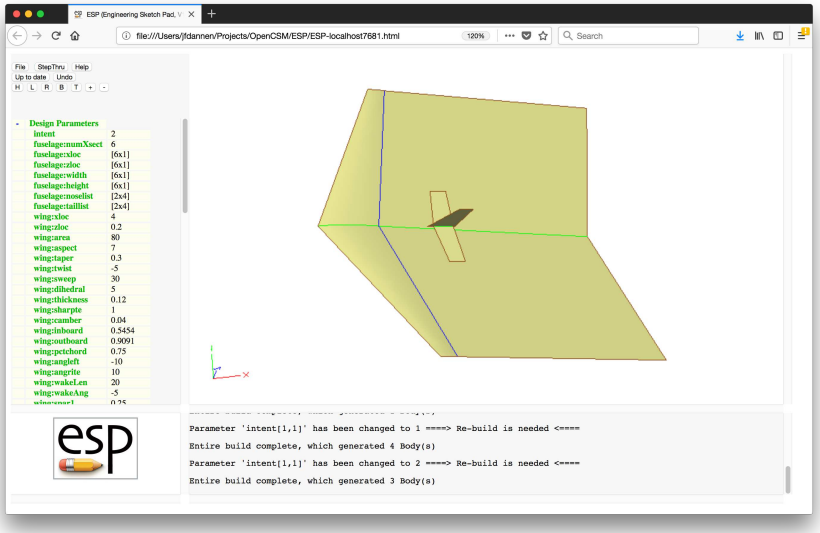
$$\text{aspect} = 7$$
$$\text{sweep} = 30$$
$$\text{taper} = 0.3$$

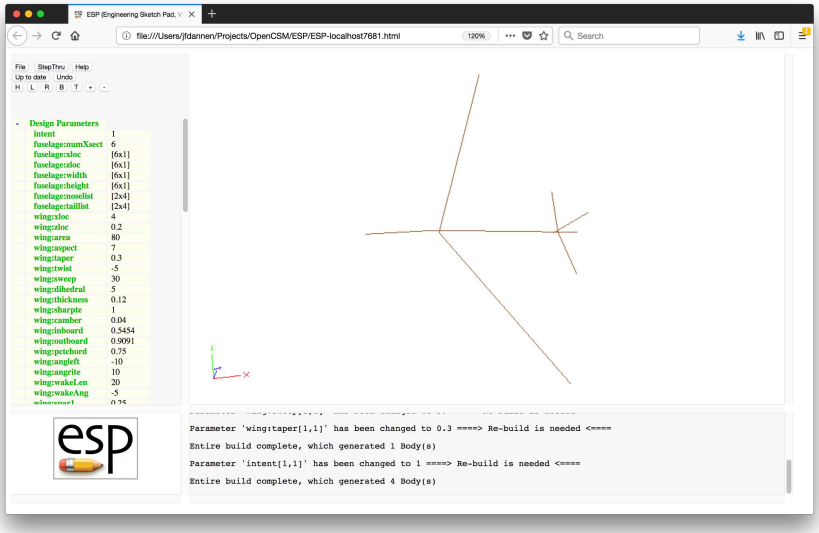
Distinguishing Features — Associative

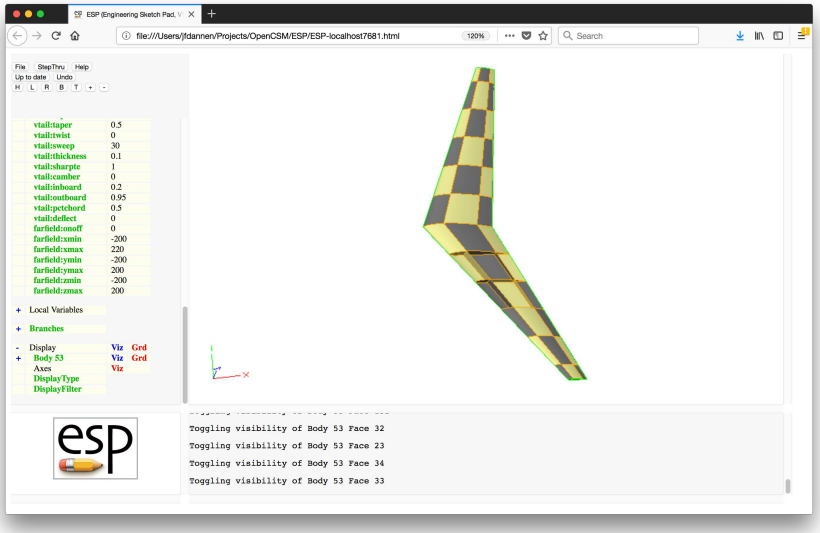
- ESP maintains a set of global and local attributes on a configuration that are persistent through rebuilds
- Supports the generation of multi-fidelity models
  - attributes can be used to associate conceptually-similar parts in the various models
- Supports the generation of multi-disciplinary models
  - 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





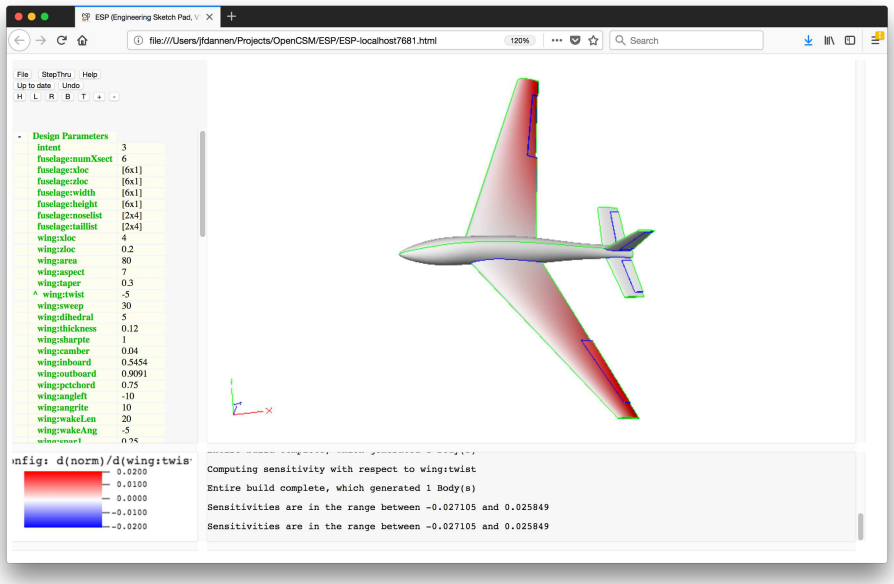




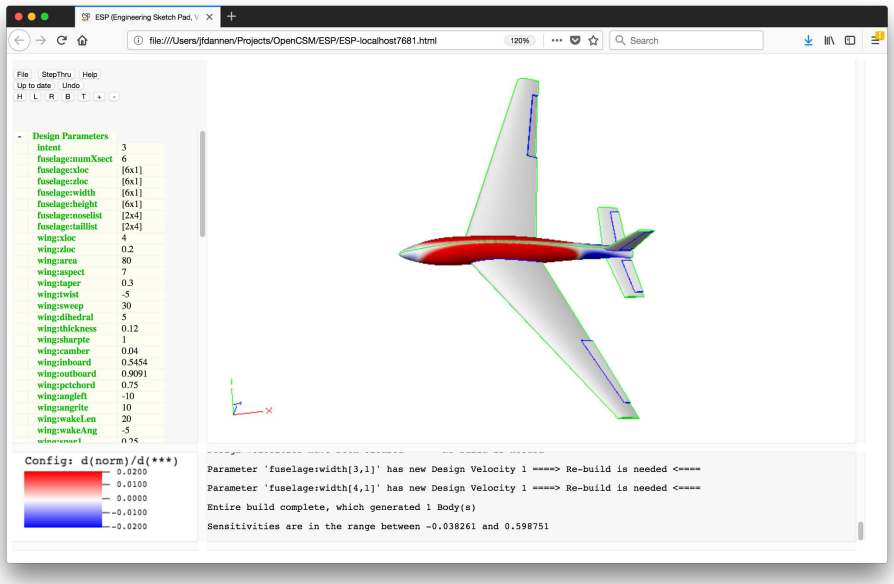
Distinguishing Features — Differentiated

- 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 `runESP115` 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 `ESP115` 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
  - `--version` or `-version` or `-v` to return version information
  - ...

ESP Command Line (2)

- Other `[options...]` include:
  - `-dict dictname` loads constants defined in dictionary file
  - `-dumpEgads` to dump `EGADS` file in form `Body_XXXXXX.egads` after each Body is built
  - `-loadEgads` to load `EGADS` file if it exists in current directory
  - `-onormal` to plot in 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)
  - `-verify` to execute `ASSERT` statements that contain `verify=1`

ESP Command Line (3)

- Other (less frequently used) `[options...]` include:
  - `-addVerify` creates verification files (for automatic regression testing)
  - `-egg eggname` uses an external grid generator
  - `-histDist dist` creates histograms of distance from plot points to the nearest surface
  - `-plot plotfile` to plot additional information
  - `-plotBDF filename` superimposes BDF information in Graphics Window
  - `-plotCP` to plot Bspline control points
  - `-sensTess` to produce configuration sensitivity (instead of tessellation sensitivity) output

ESP Screen Layout

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

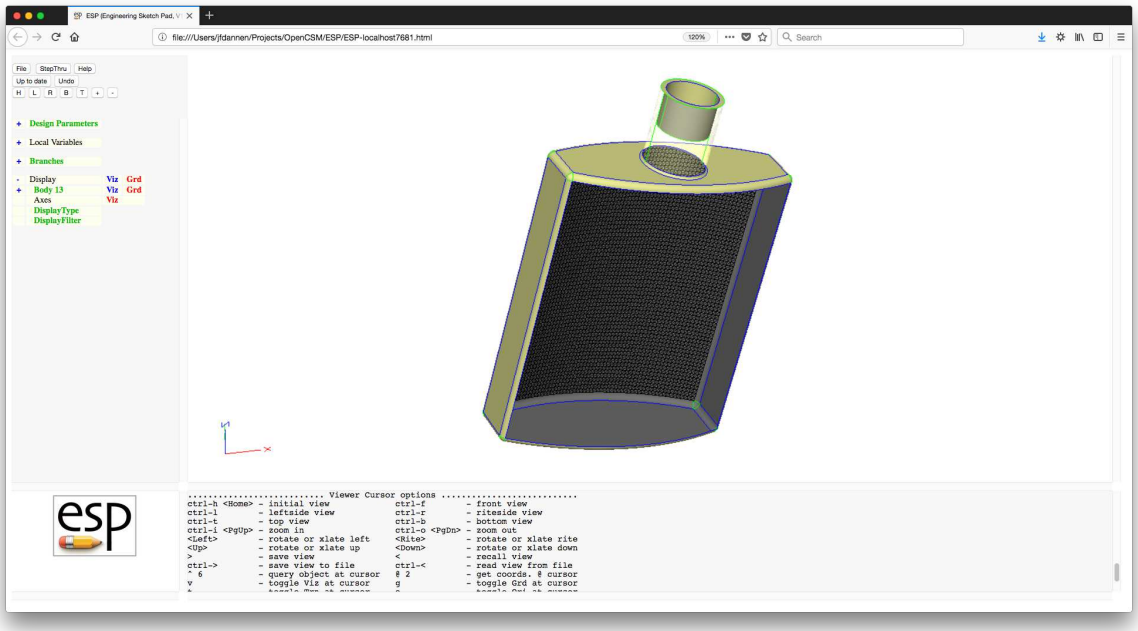


Image Manipulation via the Mouse

- 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	

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	

Button 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

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

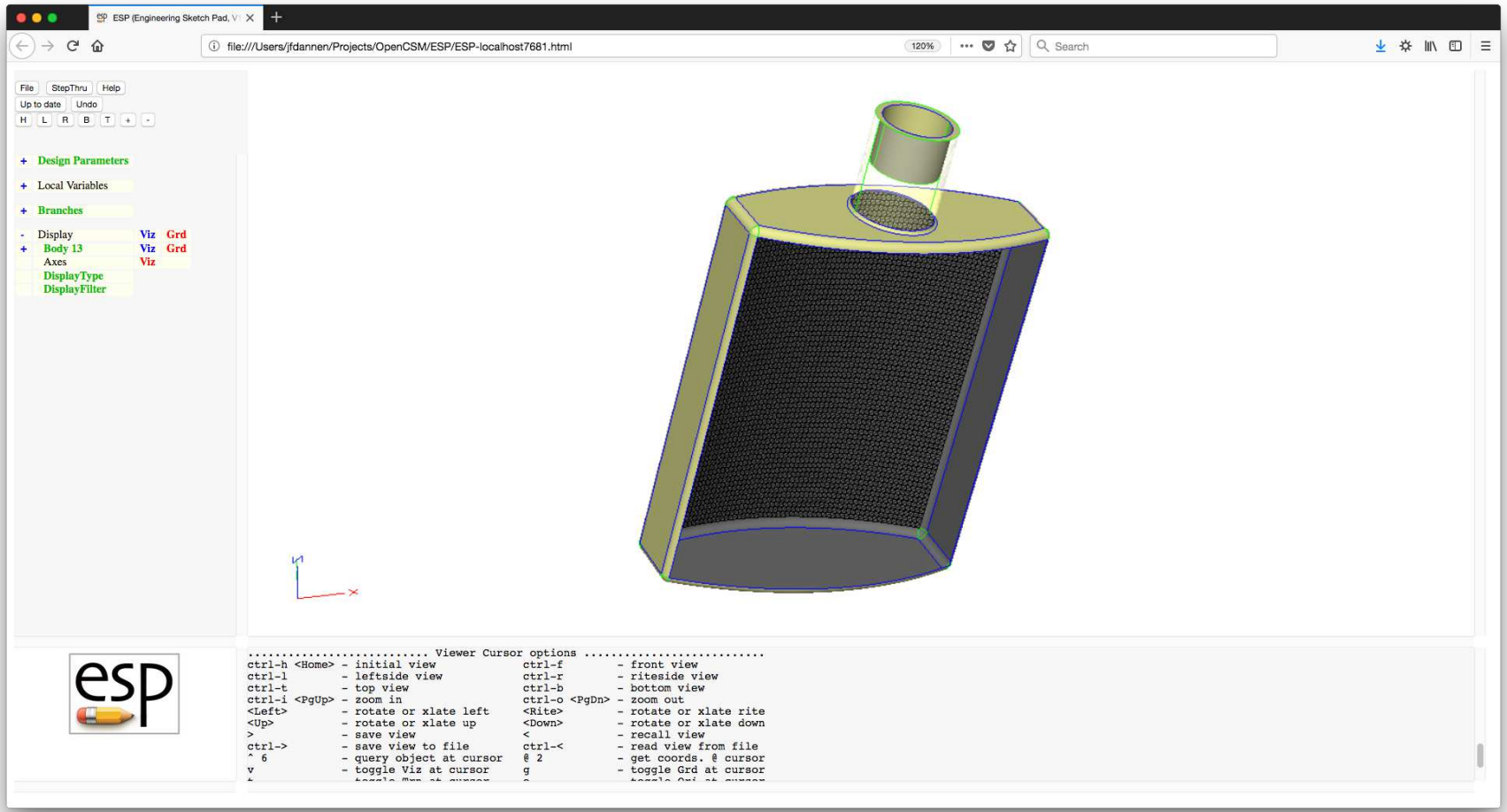
Recovering from an Error

- 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

Hands-on exercise

1. Start `serveCSM` using the file `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, ...
- Questions will be answered at next session