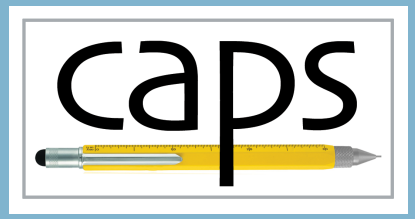
# Computational Aircraft Prototype Syntheses




## Training Session 11

## Data Transfer: Loosely-Coupled Aeroelasticity

ESP v1.19

**Marshall Galbraith**
galbramc@mit.edu

**Bob Haimes**
haimes@mit.edu

Massachusetts Institute of Technology

**John F. Dannenhoffer, III**
jfdannen@syr.edu

Syracuse University

# Overview

- Loosely coupled analysis
- `capsBound` object and the `capsBound` attribute
  - `capsVertexSet` objects
  - `capsDataSet` objects
- Loosely coupled one-way modal aeroelastic analysis
- Loosely coupled two-way iterative aeroelastic analysis
- Enhanced CAPS and Final Thoughts

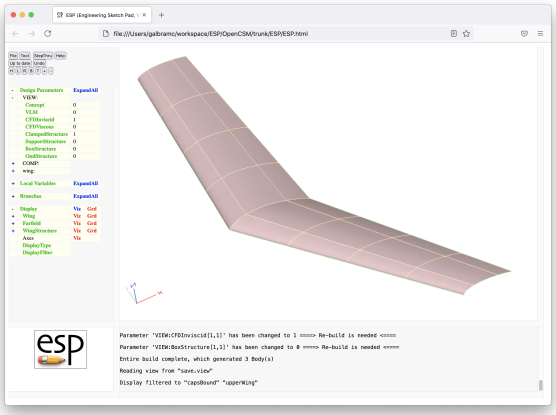
# Loosely coupled Analysis Overview

- Couple two independent analysis tools
- Aeroelastic analysis
  - CFD to compute pressures
  - FEM to compute displacements
- Typically disparate tools
  - Mesh resolution
  - Data representation (cell vs. node center)
- CAPS data transfer reconciles differences
- Examples:
  - One-way coupling: Astros Modal $\rightarrow$ Fun3D
  - Two-way coupling: Astros Static $\leftrightarrow$ SU2

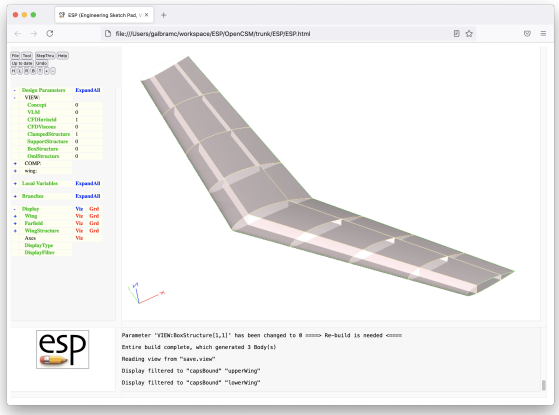
# capsBound object and attribute

- `capsBound` is a logical grouping of BRep Objects
  - Represent the same entity, e.g. "outer surface of the wing"
- Bound is used by CAPS framework to facilitate data transfer
  - Defined by the `capsBound` attribute
- Same `capsBound` attribute applied to "coincident" bodies defines connection

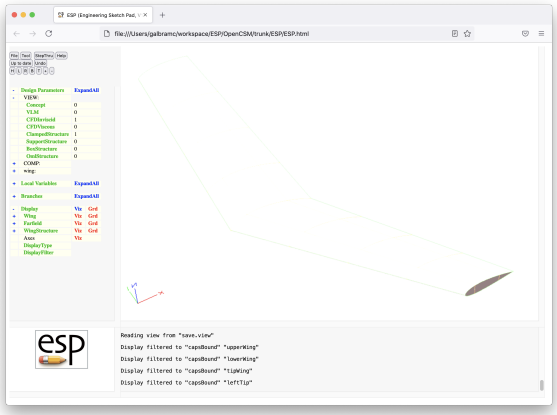
ESP/wing3.csm Structures `capsBound`



`ATTRIBUTE capsBound $upperWing`



`ATTRIBUTE capsBound $lowerWing`

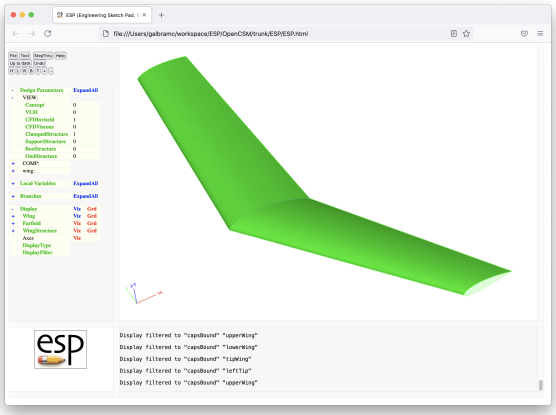


`ATTRIBUTE capsBound $leftTip`

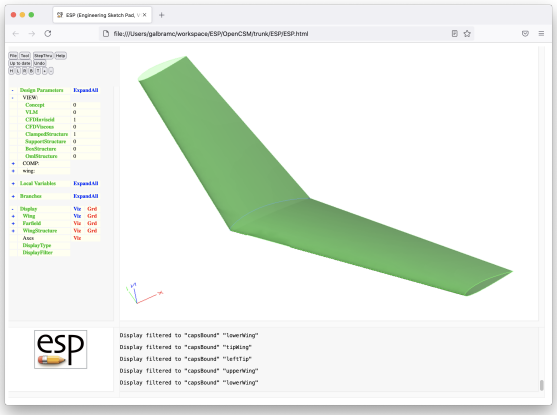
# capsBound object and attribute

- `capsBound` is a logical grouping of BRep Objects
  - Represent the same entity, e.g. "outer surface of the wing"
- Bound is used by CAPS framework to facilitate data transfer
  - Defined by the `capsBound` attribute
- Same `capsBound` attribute applied to "coincident" bodies defines connection

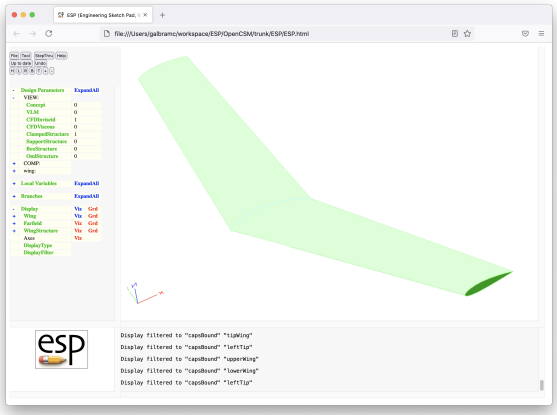
ESP/wing3.csm CFD Inviscid `capsBound`



`ATTRIBUTE capsBound $upperWing`



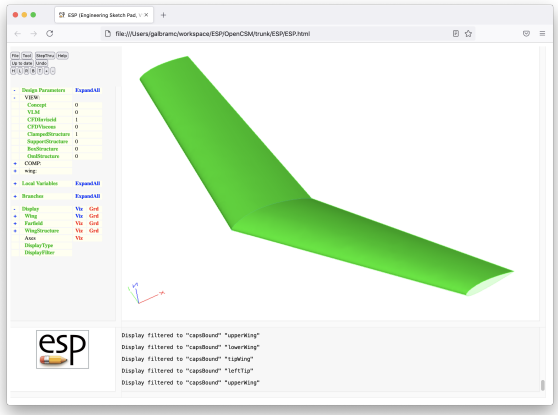
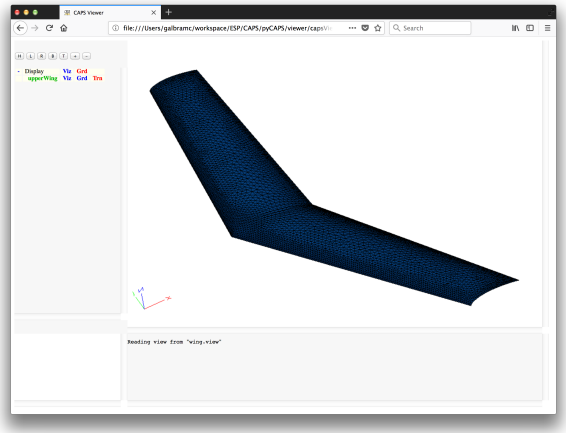
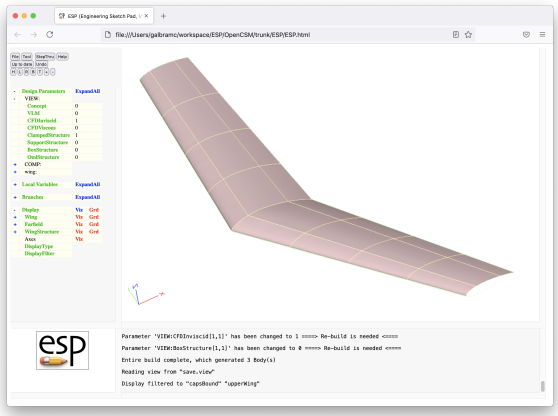
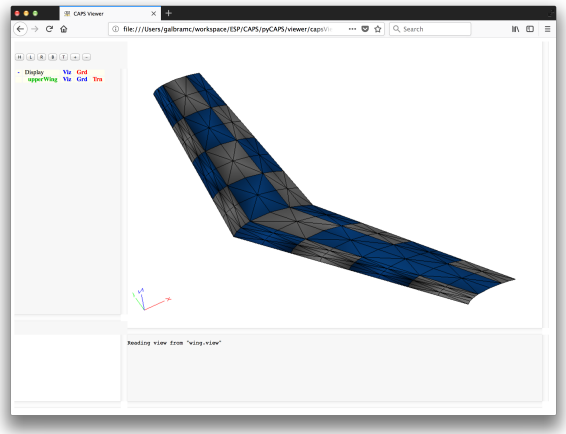
`ATTRIBUTE capsBound $lowerWing`



`ATTRIBUTE capsBound $leftTip`

# capsVertexSet object

- capsVertexSet is a discrete capsBound

| | capsBound | capsVertexSet |
|---|---|---|
| Aerodynamic |  |  |
| Structures |  |  |

# capsVertexSet object

- capsVertexSet is a discrete capsBound

- If capsBound Faces have same underlying surface, then the native UVs are used to match the points between capsVertexSets
- If not the triangulations are reparameterized with a single UV representation
- If barycentric coordinates are found for each vertex in a capsVertexSet to the other(s). This allows for straight interpolation using the solver's spatial discretization scheme (as provided in the AIM)

# capsDataSet object

## capsDataSet

- Discrete data associated with a capsVertexSet
  - Pressure
  - Temperature
  - Displacements
- Flexible data structure
  - Node centered data
  - Cell centered data
  - Higher-order nodal basis functions

# Data Transfer: Interpolate vs. Conservative

- Data transferred between different `capsVertexSet`
  - Pressure from aero to structures `capsVertexSet`
  - Displacements from structures to aero `capsVertexSet`
- Transfer via interpolation:
  - Interpolation, does not insure integrated values match between `capsVertexSets` – important for a convergent inner loop
- Conservative transfer:
  - Conservative data transfers ensure integrated quantities match by slightly adjusting (weighting) the interpolation.

# Overview

- Loosely coupled analysis
- `capsBound` object and the `capsBound` attribute
  - `capsVertexSet` objects
  - `capsDataSet` objects
- Loosely coupled one-way modal aeroelastic analysis
- Loosely coupled two-way iterative aeroelastic analysis
- Enhanced CAPS and Final Thoughts

# Example of Modal Data Transfer

- Compute EigenVectors with Astros
- Transfer EigenVectors to Fun3D for aeroelastic calculation

session11/aeroelastic_Modal_Fun3D_Astros.py

- Load aflr4AIM
- Load aflr3AIM
- Load fun3dAIM
- Load egadsTessAIM
- Load astrosAIM

- Create `capsBound` data transfers
- Generate meshes
- Fill `capsVertexSet`
- Execute ASTROS
- Transfer EigenVectors from ASTROS to Fun3D
- Execute Fun3D

# Modal Data Transfer: Create Transfer

- Create array of eigenVector names
- Create bound Object from boundName
  - capsBound: Name of the **capsBound** attribute on the bodies
- Create fun3d and astros vertex sets

session11/aeroelastic_Modal_Fun3D_Astros.py

```
# Create an array of EigenVector names
numEigenVector = 3
eigenVectors = []
for i in range(numEigenVector):
    eigenVectors.append("EigenVector_" + str(i+1))

# Create the capsBounds for data transfer
boundNames = ["upperWing", "lowerWing", "leftTip", "riteTip"]
for boundName in boundNames:
    # Create the bound
    bound = myProblem.bound.create(boundName)

    # Create the vertex sets on the bound for fun3d and astros analysis
    fun3dVset  = bound.vertexSet.create(fun3d)
    astrosVset = bound.vertexSet.create(astros)
```

# Modal Data Transfer: Create Transfer

- Create data sets for each eigenVector
- Link the data sets (interpolate or conserve)
- Close the bound

session11/aeroelastic_Modal_Fun3D_Astros.py

```
# Create the vertex sets on the bound for fun3d and astros analysis
fun3dVset  = bound.vertexSet.create(fun3d)
astrosVset = bound.vertexSet.create(astros)

# Create eigenVector data sets
for eigenVector in eigenVectors:
    fun3d_eigenVector  = fun3dVset.dataSet.create(eigenVector, pyCAPS.fType.FieldIn)
    astros_eigenVector = astrosVset.dataSet.create(eigenVector, pyCAPS.fType.FieldOut)

    # Link the data sets
    fun3d_eigenVector.link(astros_eigenVector, "Interpolate")

# Close the bound as complete (cannot create more vertex or data sets)
bound.close()
```

# Modal Data Transfer: Generate Mesh

- Generate meshes with pre/post Analysis

session11/aeroelastic_Modal_Fun3D_Astros.py

```
# Run AIM pre/post-analysis to generate the meshes
for aim in [aflr4.name, aflr3.name, tess.name]:
    myProblem.analysis[aim].preAnalysis()
    myProblem.analysis[aim].postAnalysis()
```

# Modal Data Transfer: Execution

- Execute Astros
- Set Fun3D natural frequency inputs

session11/aeroelastic_Modal_Fun3D_Astros.py

```
# Retrive natural frequencies from the structural analysis
naturalFreq = astros.output.EigenRadian # rads/s
mass        = astros.output.EigenGeneralMass

modalDict = {} # Build up Modal Aeroelastic dict
for j in eigenVectors:
    modeNum = int(j.strip("EigenVector_"))

    value = {"frequency" : naturalFreq[modeNum-1],
             "damping" : 0.99,
             "generalMass" : mass[modeNum-1],
             "generalVelocity" : 0.1}

    modalDict[j] = value

# Add Eigen information in fun3d
fun3d.input.Modal_Aeroelastic          = modalDict
```

- Execute Fun3D

# Overview

- Loosely coupled analysis
- `capsBound` object and the `capsBound` attribute
  - `capsVertexSet` objects
  - `capsDataSet` objects
- Loosely coupled one-way modal aeroelastic analysis
- Loosely coupled two-way iterative aeroelastic analysis
- Enhanced CAPS and Final Thoughts

# Example of Iterative Data Transfer

- Compute pressures with SU2
- Compute displacements with ASTROS
- Displace CFD mesh, and compute pressures with SU2

session11/aeroelastic_Iterative_SU2_Astros.py

- Load aflr4AIM
- Load aflr3AIM
- Load su2AIM
- Load egadsTessAIM
- Load astrosAIM
- Create `capsBound` Objects
- Generate meshes
- Iterate
  - Execute SU2
  - Execute ASTROS

# Iterative Data Transfer: Create Transfer

- Initial value applied to Displacement to start iterations

## session11/aeroelastic_Iterative_SU2_Astros.py

```
# Create the data transfer connections
boundNames = ["upperWing", "lowerWing", "leftTip", "riteTip"]
for boundName in boundNames:
    # Create the bound
    bound = myProblem.bound.create(boundName)

    # Create the vertex sets on the bound for su2 and astros analysis
    su2Vset    = bound.vertexSet.create(su2)
    astrosVset = bound.vertexSet.create(astros)

    # Create pressure data sets
    su2_Pressure    = su2Vset.dataSet.create("Pressure", pyCAPS.fType.FieldOut)
    astros_Pressure = astrosVset.dataSet.create("Pressure", pyCAPS.fType.FieldIn)

    # Create displacement data sets
    su2_Displacement    = su2Vset.dataSet.create("Displacement", pyCAPS.fType.FieldIn, init=[0,0,0])
    astros_Displacement = astrosVset.dataSet.create("Displacement", pyCAPS.fType.FieldOut)

    # Link the data sets
    astros_Pressure.link(su2_Pressure, "Conserve")
    su2_Displacement.link(astros_Displacement, "Interpolate")

    # Close the bound as complete (cannot create more vertex or data sets)
    bound.close()
```

# Iterative Data Transfer: Execution

- Start iterations

session11/aeroelastic_Iterative_SU2_Astros.py

```
# Aeroelastic iteration loop
for iter in range(numTransferIteration):
```

- Execute SU2

```
    # Run SU2 mesh deformation
    if iter > 0:
        su2Deform(projectName + ".cfg", numberProc)

    # Run SU2 solver
    su2Run(projectName + ".cfg", numberProc)
```

- Execute ASTROS

```
    # Run mASTROS via system call
    os.system("mastros.exe < " + projectName +  ".dat > " + projectName + ".out");
```

# Overview

- Loosely coupled analysis
- `capsBound` object and the `capsBound` attribute
  - `capsVertexSet` objects
  - `capsDataSet` objects
- Loosely coupled one-way modal aeroelastic analysis
- Loosely coupled two-way iterative aeroelastic analysis
- Enhanced CAPS and Final Thoughts

# Enhanced CAPS

## Future tasks

- Restarting the same script recycles previous data.
- Deprecate `capsIgnore` in lieu of explicit geometry removal
- Support for analysis execution
- Single UI (and integrated editor) for Geometry and Analysis

# Final Thoughts

- `ESP` is freely available for download from `acdl.mit.edu/ESP`
- Based upon user requests, new and improved features are added continually
- Send bug reports to `galbramc@mit.edu`, `haimes@mit.edu`, or `jfdannen@syr.edu`
- Also send success stories to `galbramc@mit.edu`, `haimes@mit.edu`, or `jfdannen@syr.edu`

- Thank you for attending; send comments about the course to `galbramc@mit.edu`, `haimes@mit.edu`, or `jfdannen@syr.edu`