

Computational Aircraft Prototype Syntheses



Training Session 10

Data Transfer: Loosely-Coupled Aeroelasticity

ESP v1.22

Marshall Galbraith

galbramc@mit.edu

Massachusetts Institute of Technology

Bob Haines

haines@mit.edu

John F. Dannenhoffer, III

jfdannen@syr.edu

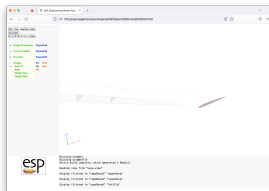
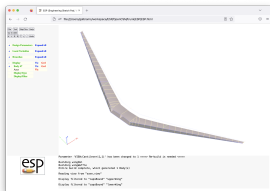
Syracuse University

- 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

- 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 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/transport.csm Structures capsBound



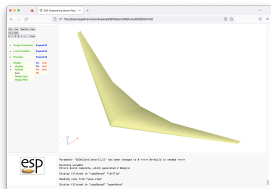
ATTRIBUTE capsBound \$upperWing

ATTRIBUTE capsBound \$lowerWing

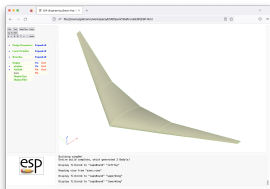
ATTRIBUTE capsBound \$leftTip

- 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

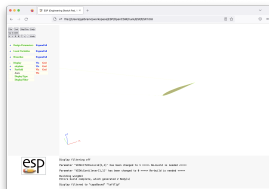
ESP/transport.csm CFD Inviscid capsBound



ATTRIBUTE capsBound \$upperWing



ATTRIBUTE capsBound \$lowerWing



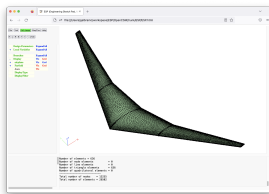
ATTRIBUTE capsBound \$leftTip

- capsVertexSet is a discrete capsBound
- Multiple VertexSets for a capsBound

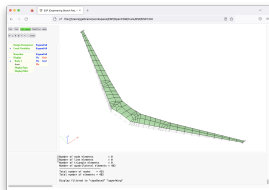
capsBound



capsVertexSet



Aerodynamic



Structures

- `capsVertexSet` is a discrete `capsBound`
 - Multiple `VertexSets` for a `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

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

- 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.

- 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

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

session10/1_aeroelastic_Modal_Fun3D_Astros.py

- Load aflr4AIM
- Load aflr3AIM
- Load fun3dAIM
- Load egadsTessAIM
- Load astrosAIM
- Create capsBound data transfers
- Execute ASTROS
- Transfer EigenVectors from ASTROS to Fun3D
- Execute Fun3D

- 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

session10/1_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 = capsProblem.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)
```

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

session10/1_aeroelastic_Modal_Fun3D_Astros.py

```
boundNames = ["upperWing", "lowerWing", "leftTip", "riteTip"]
for boundName in boundNames:
    # Create the bound
    bound = capsProblem.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)

    # Create eigenVector data sets
    for eigenVector in eigenVectors:
        fun3d_eigenVector = fun3dVset.dataSet.create(eigenVector)
        astros_eigenVector = astrosVset.dataSet.create(eigenVector)

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

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

- Execute Astros
- Set Fun3D natural frequency inputs

session10/1_aeroelastic_Modal_Fun3D_Astros.py

```
# Retrieve 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
fun3d.input.Modal_Ref_Velocity     = refVelocity
fun3d.input.Modal_Ref_Dynamic_Pressure = 0.5*refDensity*refVelocity*refVelocity
fun3d.input.Modal_Ref_Length       = 1.0
```

- Execute Fun3D

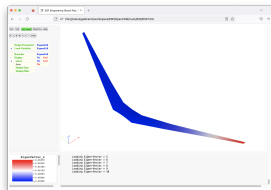
session10/1_aeroelastic_Modal_Fun3D_Astros.py

```
# Visualize capsBounds
for boundName in capsProblem.bound.keys():
    # Get the bound
    bound = capsProblem.bound[boundName]

    # Get the astros vertex set on the bound
    astrosVset = bound.vertexSet[astros.name]

    # Visualize each eigenVector data sets
    for eigenVector in eigenVectors:
        astros_eigenVector = astrosVset.dataSet[eigenVector]

        # View the data sets
        astros_eigenVector.view()
```



- 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

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

`session10/2_aeroelastic_Iterative_SU2_Astros.py`

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

- Initial value applied to Displacement to start iterations

session10/2_aeroelastic_Iterative_SU2_Astros.py

```
boundNames = ["upperWing", "lowerWing", "leftTip", "riteTip"]
for boundName in boundNames:
    # Create the bound
    bound = capsProblem.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")
    astros_Pressure = astrosVset.dataSet.create("Pressure")

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

    # 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()
```

session10/2_aeroelastic_Iterative_SU2_Astros.py

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

● Execute SU2

```
print ("\n==> Running SU2.....")
currentDirectory = os.getcwd() # Get our current working directory
os.chdir(su2.analysisDir)      # Move into test directory

numberProc = 1
# Run SU2 mesh deformation
if iter > 0:
    su2Deform(su2.input.Proj_Name + ".cfg", numberProc)

# Run SU2 solver
su2Run(su2.input.Proj_Name + ".cfg", numberProc)

shutil.copy("surface_flow_" + su2.input.Proj_Name + ".szplt", "surface_flow_" + su2.input.Proj_Name + "_Ite
os.chdir(currentDirectory) # Move back to top directory
```

● Execute ASTROS

```
# Run mASTROS via system call
print ("\n==> Running mASTROS.....")
astros.system("mastros.exe < " + astros.input.Proj_Name + ".dat > " + astros.input.Proj_Name + ".out");
```

- 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

Future tasks

- Database generation
- Full potential solve
- Metric-based Mesh adaptation for CFD (and structures)
- AIMs written in Python

- ESP is freely available for download from acd1.mit.edu/ESP
 - Based upon user requests, new and improved features are added continually
 - Send bug reports to
galbramc@mit.edu or jfdannen@syr.edu
 - Also send success stories to
galbramc@mit.edu or jfdannen@syr.edu
-
- Thank you for attending; send comments about the course to
galbramc@mit.edu or jfdannen@syr.edu