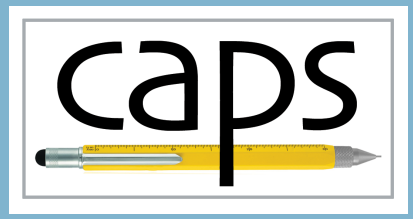
Computational Aircraft Prototype Syntheses



Training Session 8
CFD Analysis: Cart3D/SU2/Fun3D

ESP v1.22

Marshall Galbraith galbramc@mit.edu
Bob Haimes haimes@mit.edu
Massachusetts Institute of Technology

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

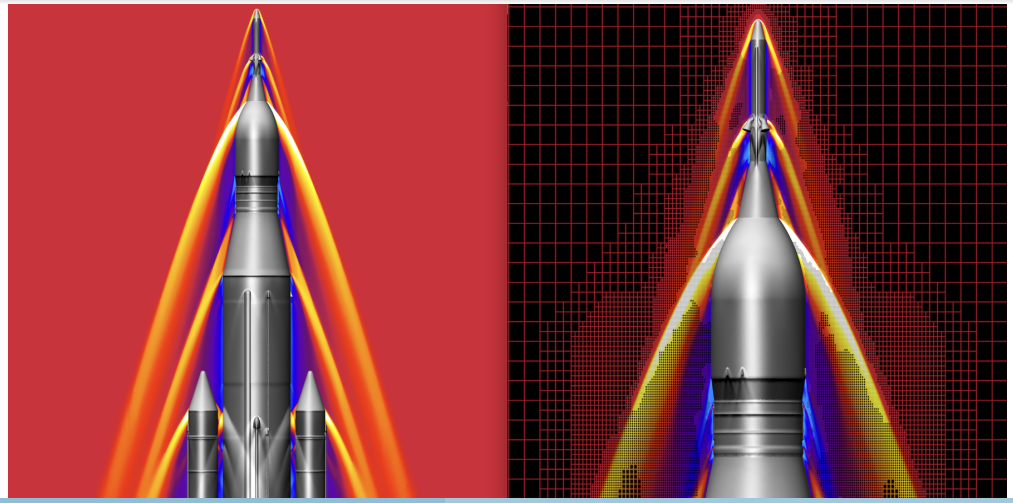
Overview

- Unstructured CFD analysis
  - Cart3D
  - SU2
  - FUN3D
- CFD analysis setup
  - CFD execution
- Optimization with CFD
  - Cart3D
  - FUN3D

Cart3D Overview

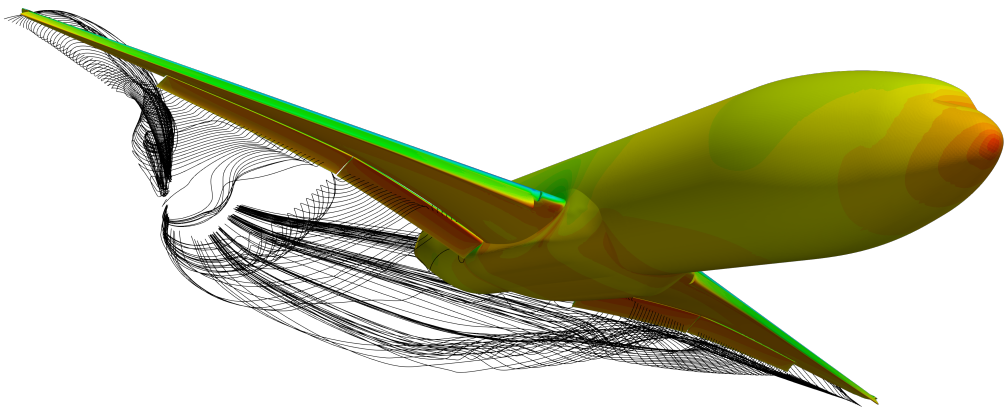
- Developed at NASA Ames

  (https://www.nas.nasa.gov/publications/software/docs/cart3d/pages/index.html)
- Nested Cartesian with immersed boundaries
- Adjoint based optimization
- CAPS: Cart3D v1.5.9 for design



SU2 Overview

- Open-source CFD solver (https://su2code.github.io/)
- Unstructured meshes
- Mesh deformation
- Adjoint based optimization
- CAPS: 4.1.1. (Cardinal), 5.0.0 (Raven), 6.2.0 (Falcon), 7.4.0 (Blackbird)

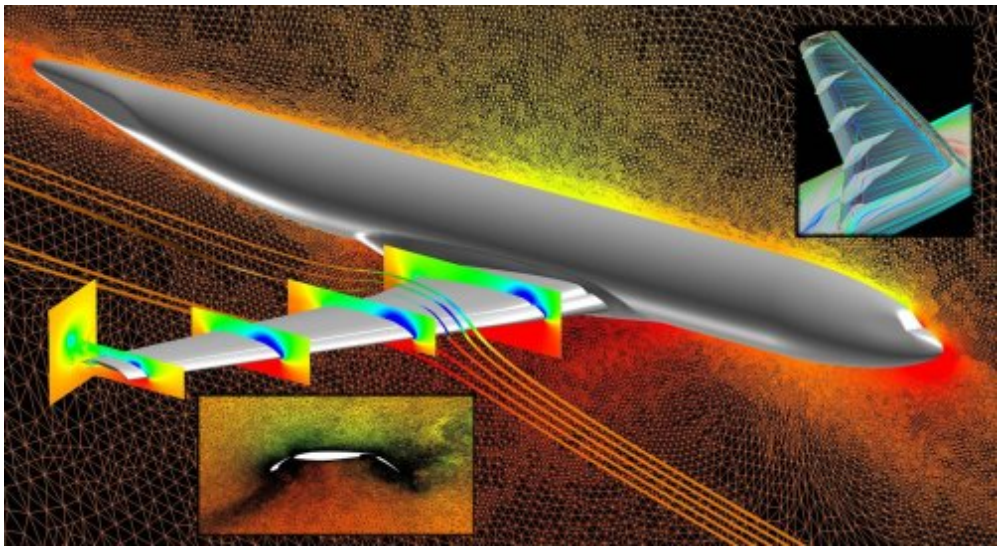


SU2

The Open-Source CFD Code

Fun3D Overview

- Developed at NASA Langley since late 1980s
- Unstructured meshes
- Mesh deformation
- Adjoint based optimization
- Much more
- CAPS: Fun3D 13.4



FUN3D
Fully Unstructured Navier-Stokes

NASA

CFD Simulation Outline

- Load geometry
- Generate mesh (the most difficult CFD input to generate)
- Create CFD AIM
  - Set CFD analysis inputs
- Execute CFD
- Extract analysis outputs

session08/1_cart3d_InviscidWing.py
session08/2_su2_InviscidWing.py
session08/3_fun3d_InviscidWing.py

Cart3D AIM Inputs and Outputs

Cart3D AIM Documentation

Cart3D AIM Analysis Inputs

session08/1_cart3d_InviscidWing.py

```
# Create Cart3D AIM
cart3d = capsProblem.analysis.create(aim = "cart3dAIM")

cart3d.geometry.view()

# Set inputs
cart3d.input.Mach          = 0.8
cart3d.input.alpha         = 1.5
cart3d.input.maxCycles     = 10
cart3d.input.nDiv          = 6
cart3d.input.maxR          = 9
cart3d.input.outer_box     = 8
cart3d.input.y_is_spanwise = True

cart3d.input.Model_X_axis = "-Xb"
cart3d.input.Model_Y_axis = "-Yb"
cart3d.input.Model_Z_axis = "-Zb"

span = transport.outpmtr.wing.span
cart3d.input.Yslices = [0, 0.5*span, span]

# Set inputs added directly to aero.csh
aerocsh = ["set it_fc = 150",
           "set it_ad = 150"]

if useMPI:
    aerocsh.append("set flowCart = mpix_flowCart")
    aerocsh.append("set mpi_prefix = 'mpiexec -n 16'")

cart3d.input.aerocsh = aerocsh
```

Cart3D Execution

- Cart3D is excued via a modified aero.csh
  - aero.csh is copied from $CART3D/bin directory

session08/2_su2_InviscidWing.py

```
# Explicitly Run Cart3D via aero.csh (optional)
cart3d.runAnalysis()

# ----------------------------------------------------------------
# Print outputs
print ("C_A   ", cart3d.output.C_A)
print ("C_Y   ", cart3d.output.C_Y)
print ("C_N   ", cart3d.output.C_N)
print ("C_D   ", cart3d.output.C_D)
print ("C_S   ", cart3d.output.C_S)
print ("C_L   ", cart3d.output.C_L)
print ("C_l   ", cart3d.output.C_l)
print ("C_m   ", cart3d.output.C_m)
print ("C_n   ", cart3d.output.C_n)
print ("C_M_x ", cart3d.output.C_M_x)
print ("C_M_y ", cart3d.output.C_M_y)
print ("C_M_z ", cart3d.output.C_M_z)
```

SU2 AIM Documentation

SU2 Mesh Generation

session08/2_su2_InviscidWing.py

```
#-----------------------------------------------------------------------------#

# Create aflr4 AIM
aflr4 = capsProblem.analysis.create(aim  = "aflr4AIM",
                                    name = "aflr4")

# Farfield growth factor
aflr4.input.ff_cdfr = 1.4

# Scaling factor to compute AFLR4 'ref_len' parameter
aflr4.input.Mesh_Length_Factor = 5

# Edge mesh spacing discontinuity scaled interpolant and farfield meshing BC
aflr4.input.Mesh_Sizing = {"leftWing": {"edgeWeight":1.0},
                           "riteWing": {"edgeWeight":1.0},
                           "Farfield": {"bcType":"Farfield"}}

#-----------------------------------------------------------------------------#

# Create AFLR3 AIM to generate the volume mesh
aflr3 = capsProblem.analysis.create(aim  = "aflr3AIM",
                                    name = "aflr3")

# Link the aflr4 Surface_Mesh as input to aflr3
aflr3.input["Surface_Mesh"].link(aflr4.output["Surface_Mesh"])

#-----------------------------------------------------------------------------#
```

SU2 AIM Analysis Inputs

session08/2_su2_InviscidWing.py

```
# Link the aflr3 Volume_Mesh as input to su2
su2.input["Mesh"].link(aflr3.output["Volume_Mesh"])

# Set project name. Files written to analysisDir will have this name
su2.input.Proj_Name = "inviscidWing"

su2.input.Alpha = 1.0                    # AoA
su2.input.Mach = 0.5                     # Mach number
su2.input.Equation_Type = "Compressible" # Equation type
su2.input.Num_Iter = 5                   # Number of iterations

# Set number of inner iterations and convergence
su2.input.Input_String = ["INNER_ITER= 10",
                          "CONV_FIELD= RMS_DENSITY",
                          "CONV_RESIDUAL_MINVAL= -8"]

# Set boundary conditions via capsGroup
inviscidBC = {"bcType" : "Inviscid"}
su2.input.Boundary_Condition = {"Wing"     : inviscidBC,
                                "Farfield": "farfield"}

# Specifcy the boundares used to compute forces
su2.input.Surface_Monitor = ["Wing"]

# Set SU2 Version
su2.input.SU2_Version = "Blackbird"
```

SU2 Execution

- Execute using SU2 python interface

session08/2_su2_InviscidWing.py

```
# Import SU2 python environment
from parallel_computation import parallel_computation as su2Run
```

```
# Run AIM pre-analysis
su2.preAnalysis()

####### Run SU2 ########################
print ("\n\nRunning SU2......")
currentDirectory = os.getcwd() # Get our current working directory

os.chdir(su2.analysisDir) # Move into test directory

# Run SU2 with specified number of partitions
su2Run(su2.input.Proj_Name + ".cfg", partitions = 1)

os.chdir(currentDirectory) # Move back to top directory
########################################

# Run AIM post-analysis
su2.postAnalysis()
```

SU2 Extracting Analysis Output

- Retrieve forces and moments

session08/2_su2_InviscidWing.py

```
# Get Lift and Drag coefficients
print ("--> Cl = ", su2.output.CLtot,
           "Cd = ", su2.output.CDtot)

# Get Cmx, Cmy, and Cmz coefficients
print ("--> Cmx = ", su2.output.CMXtot,
           "Cmy = ", su2.output.CMYtot,
           "Cmz = ", su2.output.CMZtot)

# Get Cx, Cy, Cz coefficients
print ("--> Cx = ", su2.output.CXtot,
           "Cy = ", su2.output.CYtot,
           "Cz = ", su2.output.CZtot)
```

Fun3D AIM Inputs and Outputs

Fun3D AIM Documentation

Fun3D AIM Analysis Inputs

session08/3_fun3d_InviscidWing.py

```
# Link the aflr3 Volume_Mesh as input to fun3d
fun3d.input["Mesh"].link(aflr3.output["Volume_Mesh"])

# Set project name. Files written to analysisDir will have this name
fun3d.input.Proj_Name = "inviscidWing"

fun3d.input.Alpha = 1.0                      # AoA
fun3d.input.Mach = 0.5                       # Mach number
fun3d.input.Equation_Type = "Compressible" # Equation type
fun3d.input.Num_Iter = 5                     # Number of iterations
fun3d.input.Restart_Read = 'off'             # Do not read restart
fun3d.input.Viscous = "inviscid"             # Inviscid calculation

# Set boundary conditions via capsGroup
inviscidBC = {"bcType" : "Inviscid"}
fun3d.input.Boundary_Condition = {"Wing"    : inviscidBC,
                                  "Farfield":"farfield"}

# Use python to add inputs to fun3d.nml file
fun3d.input.Use_Python_NML = True
```

Fun3D NML Analysis Inputs

- fun3d.nml is very large (and changes with Fun3D versions)
- Not all inputs implemented in AIM
- f90nml used to write directly to fun3d.nml
  - NOTE: Circumvents CLEAN/DIRTY process
  - Always use AIM inputs when available

session08/3_fun3d_InviscidWing.py

```
# Use python to add inputs to fun3d.nml file
fun3d.input.Use_Python_NML = True

# Write boundary output variables to the fun3d.nml file directly
fun3dnml = f90nml.Namelist()
fun3dnml['boundary_output_variables'] = f90nml.Namelist()
fun3dnml['boundary_output_variables']['mach'] = True
fun3dnml['boundary_output_variables']['cp'] = True
fun3dnml['boundary_output_variables']['average_velocity'] = True

fun3dnml.write(os.path.join(fun3d.analysisDir,"fun3d.nml"), force=True)
```

Fun3D Execution

- Execute Fun3D using system call

session08/3_fun3d_InviscidWing.py

```
# Run AIM pre-analysis
fun3d.preAnalysis()

print ("\n==> Running FUN3D......")
# Run fun3d via system call
fun3d.system("nodet_mpi --animation_freq -1 --write_aero_loads_to_file > Info.out")

# Run AIM post-analysis
fun3d.postAnalysis()
```

Fun3D Extracting Analysis Output

- Retrieve forces and moments

session08/3_fun3d_InviscidWing.py

```
# Get force results
print ("\n==> Total Forces and Moments")
# Get Lift and Drag coefficients
print ("--> Cl = ", fun3d.output.CLtot,
           "Cd = ", fun3d.output.CDtot)

# Get Cmx, Cmy, and Cmz coefficients
print ("--> Cmx = ", fun3d.output.CMXtot,
           "Cmy = ", fun3d.output.CMYtot,
           "Cmz = ", fun3d.output.CMZtot)

# Get Cx, Cy, Cz coefficients
print ("--> Cx = ", fun3d.output.CXtot,
           "Cy = ", fun3d.output.CYtot,
           "Cz = ", fun3d.output.CZtot)
```

Overview

- Unstructured CFD analysis
  - Cart3D
  - SU2
  - FUN3D
- CFD analysis setup
  - CFD execution
- Optimization with CFD
  - Cart3D
  - FUN3D

CAPS+Cart3D Development

- Collaborative effort by
  - Marshall Galbraith and Bob Haimes (MIT)
  - Marian Nemec and Micheal Aftosmis (NASA Ames)
- New features added to CAPS framework and Cart3D AIM
- Cart3D design framework scripts updated
  - c3d_objGrad.csh updated to support execution outside Cart3D design framework
  - Updated xddm C-library for Cart3D design xml files
- Updated Cart3D design framework released with v1.5.9

Two Example Optimization Problems

- Minimize C_L^2
  - Analysis Input α design variable
  - Geometry Input wing:alphar (root twist) design variables

Optimization with Analysis Input Sensitivity

- Gradient based optimization

$$\min_{\alpha} \; C_L^2 \tag{1}$$

- Transport Hershey-bar wing with symmetric airfoil
  - Exact solution: $\alpha = 0$
- Use Cart3D to compute C_L^2 and $\partial C_L^2 / \partial \alpha$

Cart3D AIM Design Analysis Inputs

- Design analysis inputs
  - Design_Functional: Functionals defined in Cart3D design.xml
  - Design_Variable: Variables for partial derivatives

session08/4_cart3d_OpenMDAO_alpha.py

```
# Setup Cl**2 as an output functional
cart3d.input.Design_Functional = {"CL2": {"function":"C_L", "power":2}}

# Declare design variables
cart3d.input.Design_Variable = {"alpha":""}
```

```
cart3d.input.alpha          = 1.5
```

OpenMDAO Component: Setup

- Define output functional and partial derivatives for OpenMDAO

session08/4_cart3d_OpenMDAO_alpha.py

```
    def setup(self):
        # Assign the actual values to the variables.

        capsProblem = self.options['capsProblem']
        cart3d = capsProblem.analysis['cart3d']

        # Add free design parameters to OpenMDAO object
        self.add_input('alpha', val=cart3d.input.alpha)

        # Add outputs
        self.add_output('CL2')

        # Add partials
        self.declare_partials('CL2', 'alpha')
```

OpenMDAO Component: Compute

- Compute the functional C_L^2
- Set *alpha* based on "inputs" from OpenMDAO
- Disable Design_Sensitvity
- Dynamic Output (dynout) defined by Design_Functional

session08/4_cart3d_OpenMDAO_alpha.py

```
    def compute(self, inputs, outputs):
        # Compute functionals

        capsProblem = self.options['capsProblem']
        cart3d = capsProblem.analysis['cart3d']

        # Update input values
        if cart3d.input.alpha != inputs['alpha']:
            cart3d.input.alpha = inputs['alpha']

        # Disable design sensitivity calculations
        if cart3d.input.Design_Sensitivity != False:
            cart3d.input.Design_Sensitivity = False

        # Return the functional output (triggers autoExecution of c3d_objGrad.csh)
        outputs['CL2'] = cart3d.dynout["CL2"].value
```

```
cart3d.input.Design_Functional = {"CL2": {"function":"C_L", "power":2}}
```

Cart3D AIM Auto-execution

- Cart3D AIM generates all inputs required for Design_Functional evaluation

```
cart3d.analysisDir
├── inputs
│   ├── aero.csh autoInputs.out Components.i.tri Config.xml
│   └── input.c3d input.cntl preSpec.c3d.cntl
└── design
    └── OPTIONS design.xml builder.xml
```

- Cart3D AIM executes c3d_objGrad.csh
- Functionals read from updated design.xml

OpenMDAO Component: Compute Partials

- Compute the partial $\partial C_L^2/\partial\alpha$
- Enable Design_Sensitvity

session08/4_cart3d_OpenMDAO_alpha.py

```
    def compute_partials(self,inputs,partials):
        # Compute partial derivatives of the functionals

        capsProblem = self.options['capsProblem']
        cart3d = capsProblem.analysis['cart3d']

        # Update input values
        if cart3d.input.alpha != inputs['alpha']:
            cart3d.input.alpha = inputs['alpha']

        # Compute analysis and/or geometric sensitivities for the Design_Functional
        if cart3d.input.Design_Sensitivity != True:
            cart3d.input.Design_Sensitivity = True

        # Get derivatives and set partials (triggers autoExecution of c3d_objGrad.csh)
        partials['CL2', 'alpha']  = cart3d.dynout["CL2"].deriv("alpha")
```

```
cart3d.input.Design_Variable = {"alpha":""}
```

- c3d_objGrad.csh "restarts" and only computes partial derivative (via adjoint) and updates design.xml

OpenMDAO: Execution

session08/4_cart3d_OpenMDAO_alpha.py

```
# Setup the OpenMDAO problem object
omProblem = om.Problem()

# Add the fluid analysis - all the options are added as arguments
cart3dSystem = Cart3dAnalysis(capsProblem = capsProblem)

# setup the optimization
omProblem.driver = om.ScipyOptimizeDriver()
omProblem.driver.options['optimizer'] = "L-BFGS-B"
omProblem.driver.options['tol'] = 1.e-6
omProblem.driver.options['disp'] = True
omProblem.driver.options['maxiter'] = 10

# Add the Cart3D subsystem
omProblem.model.add_subsystem('cart3dSystem', cart3dSystem)

# Add design variables to model
omProblem.model.add_design_var('cart3dSystem.alpha', lower=-10.0, upper=10.0)

# Set objective function to minimize Cl**2
omProblem.model.add_objective('cart3dSystem.CL2')

# Press go
omProblem.setup()
omProblem.run_driver()
omProblem.cleanup()

# Get the optimized value
opt_alpha = omProblem.get_val("cart3dSystem.alpha")
print("Optimized alpha:", opt_alpha)
```

Cart3D Optimization with Geometric Sensitivities

- Gradient based optimization

$$\min_{\text{wing:alphar}} \quad C_L^2 \tag{2}$$

- Hershey-bar wing with symmetric airfoil
  - Twist angle defined at root airfoil section
  - Exact solution: wing : alphar $= 0$
- Use Cart3D to compute C_L^2 and $\partial C_L^2 / \partial$wing : alphar
  - CAPS computes geometric sensitivities

Cart3D AIM Design Analysis Inputs

- Design analysis inputs
  - Design_Functional: Functionals defined in Cart3D design.xml
  - Design_Variable: Variables for partial derivatives

session08/5_cart3d_OpenMDAO_twist.py

```
# Setup Cl**2 as an output functional
cart3d.input.Design_Functional = {"CL2": {"function":"C_L", "power":2}}

# Declare design variables
cart3d.input.Design_Variable = {"wing:alphar":""}
```

ESP/wingPmtrs.udc

```
DESPMTR   wing:alphar      -1.0    # setting angle   at root
```

OpenMDAO Component: Setup

- Define output functional and partial derivatives for OpenMDAO

session08/5_cart3d_OpenMDAO_twist.py

```
    def setup(self):
        # Assign the actual values to the variables.

        capsProblem = self.options['capsProblem']
        geometry = capsProblem.geometry

        # Add free design parameters to OpenMDAO object
        self.add_input('wing:alphar', val=geometry.despmtr.wing.alphar)

        # Add outputs
        self.add_output('CL2')

        # Add partials
        self.declare_partials('CL2','wing:alphar')
```

OpenMDAO Component: Compute

- Compute the functional C_L^2
- Set wing:alphar based on "inputs" from OpenMDAO
- Disable Design_Sensitvity

session08/5_cart3d_OpenMDAO_twist.py

```
    def compute(self, inputs, outputs):
        # Compute functionals

        capsProblem = self.options['capsProblem']
        cart3d = capsProblem.analysis['cart3d']

        # Update input values
        if capsProblem.geometry.despmtr.wing.alphar != inputs['wing:alphar']:
            capsProblem.geometry.despmtr.wing.alphar = inputs['wing:alphar']

        # Disable design sensitivity calculations
        if cart3d.input.Design_Sensitivity != False:
            cart3d.input.Design_Sensitivity = False

        # Return the functional output (triggers autoExecution of c3d_objGrad.csh)
        outputs['CL2'] = cart3d.dynout["CL2"].value
```

```
cart3d.input.Design_Functional = {"CL2": {"function":"C_L", "power":2}}
```

OpenMDAO Component: Compute Partials

- Compute the partial $\partial C_L^2/\partial$wing : alphar
- Enable Design_Sensitivity

session08/5_cart3d_OpenMDAO_twist.py

```
    def compute_partials(self,inputs,partials):
        # Compute partial derivatives of the functionals

        capsProblem = self.options['capsProblem']
        cart3d = capsProblem.analysis['cart3d']

        # Update input values
        if cart3d.input.alpha != inputs['alpha']:
            cart3d.input.alpha = inputs['alpha']

        # Compute analysis and/or geometric sensitivities for the Design_Functional
        if cart3d.input.Design_Sensitivity != True:
            cart3d.input.Design_Sensitivity = True

        # Get derivatives and set partials (triggers autoExecution of c3d_objGrad.csh)
        partials['CL2', 'wing:alphar']  = cart3d.dynout["CL2"].deriv("wing:alphar")
```

```
cart3d.input.Design_Variable = {"wing:alphar":""}
```

Cart3D AIM Geometric Sensitivity

- Cart3D AIM generates all tessellation sensitivity files

```
cart3d.analysisDir
└── design
    └── geometry_CAPSmodel
        └── geometry
            └── Model__CAPSmodel__Variable__wing:alphar_1_1_
                └── Components.i.tri
```

- Cart3D AIM executes c3d_objGrad.csh “restart ” via system call
- OpenMDAO execution the same as before

Fun3D Optimization with Constraint

- Analysis and Geometric sensitivities
- Gradient based optimization

$$\min_{\alpha,\text{wing:aspect}} \quad C_D^2 \tag{3}$$

$$s.t.\ C_L \equiv 0.1 \tag{4}$$

- Use Fun3D to compute C_D^2, C_L and $\partial/\partial\alpha$, $\partial/\partial aspect$
  - CAPS computes geometric sensitivities
- New mesh generated with each shape change
  - Support large deformation and topological changes

Fun3D AIM Design Analysis Inputs

- Design analysis inputs
  - Design_Functional: Functionals in Fun3D rubber.data
  - Design_Variable: Variables for partial derivatives

session08/6_fun3d_OpenMDAO_remesh.py

```
# Setup Cd**2 as the objective functional to minimize
# and Cl for the constraint
fun3d.input.Design_Functional = {"Cd2": {"function":"Cd", "power":2},
                                 "Cl" : {"function":"Cl"}}

# Declare design variables
fun3d.input.Design_Variable = {"Alpha":"",
                               "wing:aspect":""}
```

```
fun3d.input.Alpha = 1.0                    # AoA
```

ESP/wingPmtrs.udc

```
DESPMTR   wing:aspect      9.00    # aspect ratio
```

OpenMDAO Component: Setup

- Define output functional and partial derivatives for OpenMDAO

session08/6_fun3d_OpenMDAO_remesh.py

```
    def setup(self):
        # Assign the actual values to the variables.
        capsProblem = self.options['capsProblem']
        fun3d = capsProblem.analysis['fun3d']

        # attach free design parameters to OpenMDAO fluid object
        self.add_input('Alpha' , val=fun3d.input.Alpha)
        self.add_input('wing:aspect', val=capsProblem.geometry.despmtr.wing:aspect)

        # Add outputs for both objective and constraints
        self.add_output('Cd2')
        self.add_output('Cl')

        # Add partials
        self.declare_partials('Cd2','Alpha')
        self.declare_partials('Cd2','wing:aspect')

        self.declare_partials('Cl','Alpha')
        self.declare_partials('Cl','wing:aspect')
```

OpenMDAO Component: Compute

- Disable Design_Sensitvity, run nodet_mpi

session08/6_fun3d_OpenMDAO_remesh.py

```
    def compute(self, inputs, outputs):
        # Compute functionals
        cores = self.options['cores']
        capsProblem = self.options['capsProblem']
        fun3d = capsProblem.analysis['fun3d']

        # Update input values
        fun3d.input.Alpha = inputs['Alpha']
        capsProblem.geometry.despmtr.wing.aspect = inputs['wing:aspect']

        # Disable design sensitivity calculations
        fun3d.input.Design_Sensitivity = False

        # Run forward analysis
        fun3d.preAnalysis()
        fun3d.system(f"mpirun -np {cores} nodet_mpi --design_run | tee nodet.out", 'Flow')
        fun3d.postAnalysis()

        # Grab functional and attach as an output
        outputs['Cd2'] = fun3d.dynout.Cd2
        outputs['Cl']  = fun3d.dynout.Cl
```

```
fun3d.input.Design_Functional = {"Cd2": {"function":"Cd", "power":2},
                                 "Cl" : {"function":"Cl"}}
```

OpenMDAO Component: Compute Partials

- Enable Design_Sensitivity

session08/6_fun3d_OpenMDAO_remesh.py

```
    def compute_partials(self,inputs,partials):
        # The actual value of the partial derivative is assigned here.
        cores = self.options['cores']
        capsProblem = self.options['capsProblem']
        fun3d = capsProblem.analysis['fun3d']

        # Request analysis and/or geometric design sensitivities for the Design_Functional
        fun3d.input.Design_Sensitivity = True

        # Run adjoint
        fun3d.preAnalysis()
        fun3d.system(f"mpirun -np {cores} dual_mpi --getgrad --outer_loop_krylov | tee dual.out",'Adjoint')
        fun3d.postAnalysis()

        # Get derivatives and set partials
        partials['Cd2', 'Alpha'      ]  = fun3d.dynout["Cd2"].deriv("Alpha")
        partials['Cd2', 'wing:aspect'] = fun3d.dynout["Cd2"].deriv("wing:aspect")

        partials['Const', 'Alpha'      ]  = fun3d.dynout["Cl"].deriv("Alpha")
        partials['Const', 'wing:aspect'] = fun3d.dynout["Cl"].deriv("wing:aspect")
```

```
fun3d.input.Design_Variable = {"Alpha":"",
                               "wing:aspect":""}
```

OpenMDAO: Execution

session08/6_fun3d_OpenMDAO_remesh.py

```
# setup the optimization
omProblem.driver = om.ScipyOptimizeDriver()
omProblem.driver.options['optimizer'] = 'SLSQP'
#omProblem.driver.options['optimizer'] = "L-BFGS-B"
omProblem.driver.options['tol'] = 1.e-7
omProblem.driver.options['disp'] = True
omProblem.driver.options['maxiter'] = 10

# add design variables to model
aspect_limits = transport.despmtr["wing:aspect"].limits

omProblem.model.add_design_var('fun3dSystem.Alpha' , lower=-25.0, upper=25.0)
omProblem.model.add_design_var('fun3dSystem.wing:aspect', lower=aspect_limits[0], upper=aspect_limits[1])

# minimize Cd**2 subject to Cl == 0.1
omProblem.model.add_objective('fun3dSystem.Cd2')
omProblem.model.add_constraint('fun3dSystem.Cl', equals = 0.1)

# Press go
omProblem.setup()
omProblem.run_driver()
omProblem.cleanup()

# Get the optimized value
opt_Cd2    = omProblem.get_val("fun3dSystem.Cd2")
opt_Cl     = omProblem.get_val("fun3dSystem.Cl")
opt_Alpha  = omProblem.get_val("fun3dSystem.Alpha")
opt_aspect = omProblem.get_val("fun3dSystem.wing:aspect")
```

Fun3D Optimization with Mesh Morphing

- Mesh morphing might have smoother sensitivities
- Limited to smaller deformations
- No geometric topological changes

session08/7_fun3d_OpenMDAO_morph.py

```
# Enable Mesh_Morph to save off a reference mesh
fun3d.input.Mesh_Morph = True
```

- Compute with additional arguments
- Unlink mesh to dissable re-generation

```
# Run forward analysis
fun3d.preAnalysis()
fun3d.system(f"mpirun -np {cores} nodet_mpi --design_run " +
             f"--read_surface_from_file --write_mesh {fun3d.input.Proj_Name} | tee nodet.out", 'Flow')
fun3d.postAnalysis()

# Grab functional and attach as an output
outputs['Cd2'] = fun3d.dynout.Cd2
outputs['Cl']  = fun3d.dynout.Cl

# Unlink the mesh to trigger morphing
fun3d.input["Mesh"].unlink()
```

Suggested Exercises

- Create your own (optionally share it `galbramc@mit.edu`)