# FUN3D v12.7 Training

# Session 10:
# Feature- and Adjoint-Based Error Estimation and Mesh Adaptation

Mike Park



## Learning Goals

- Background on adaptation
- Manual step-by-step output adaptation cycle
- Describe the scripts that automate this process

# Available Adaptation Modes

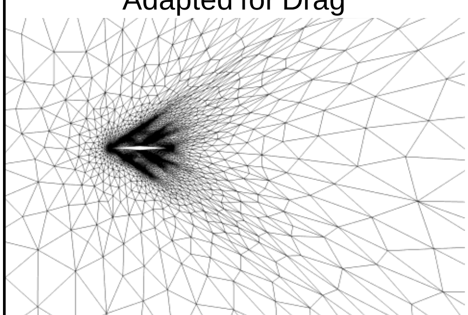
- Split into error-estimation/metric construction and adaptive mechanics
- Output-based adaptation for capabilities with an adjoint
- Local-error or feature-based adaptation for other flow solver capabilities
- Anisotropic metric-based triangular and tetrahedral grid adaptation with a frozen mixed element boundary layer that can be subdivided
- Experimental grid adaptation for time accurate simulations
- Controlled with the `&adapt_mechanics` and `&adapt_metric_construction` namelists
- See FUN3D user manual grid adaptation overview section and complete namelist description



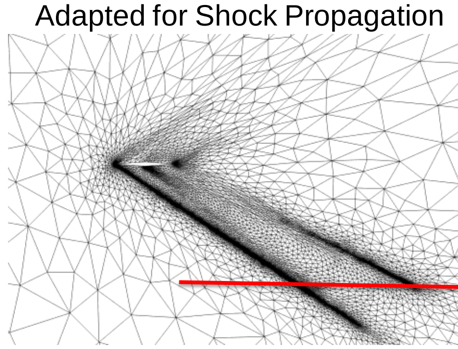
# Output-Based Adaptation

- Mathematically rigorous approach involving the adjoint solution that reduces estimated error in an engineering output
- Uniformly reducing discretization error is not ideal from an engineering standpoint - some errors are more important to outputs

Adapted for Drag

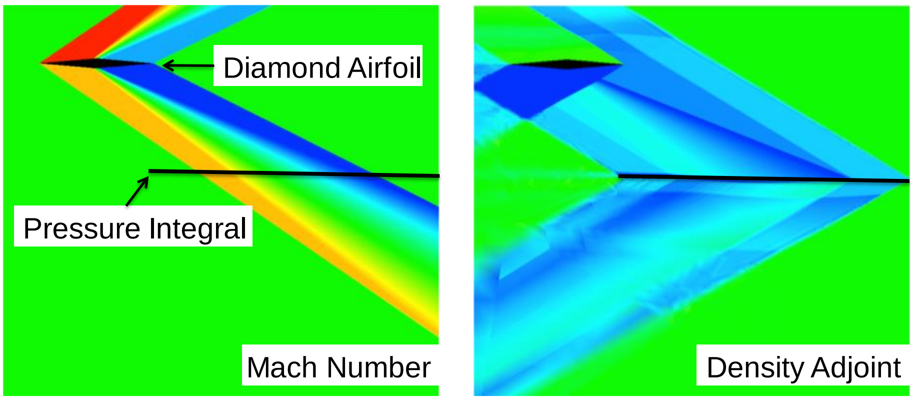


Adapted for Shock Propagation



## Shock Propagation Example

- Adaptation is targeted to improve off-body pressure integral output for diamond airfoil






## Local Error and Output Adaptation

**Local error based**

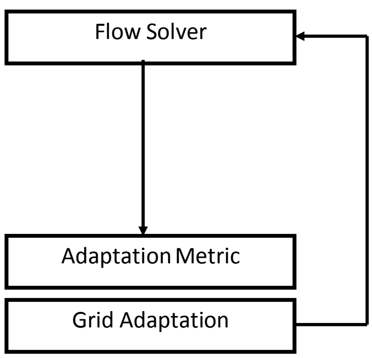
- Feature based adaptation
- Flow solver/physics agnostic
- Not as robust
- Requires more manual interaction

**Output (adjoint) based**

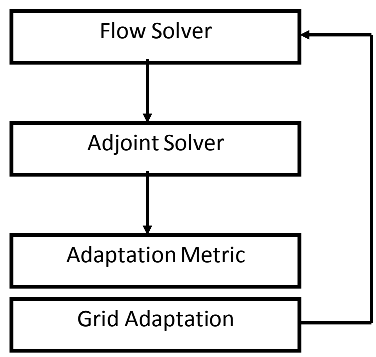
- Requires adjoint solution
- More robust
- Transport of errors
- Fewer user controlled parameters

# Adaptation Process

**Local error based**



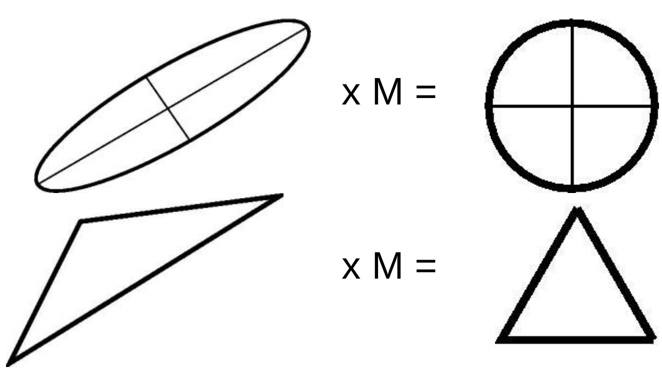

**Output (adjoint) based**






# Metric Adaptation Mechanics

- Parallel node insertion, node movement, element collapse, and element swap to iteratively drive mesh to satisfy an anisotropic metric M



# Metric

- Eigenvalue decomposition of the metric reveals a spacing request in a rotated orthogonal basis

$$X \begin{bmatrix} \left(\frac{1}{h_1}\right)^2 & & \\ & \left(\frac{1}{h_2}\right)^2 & \\ & & \left(\frac{1}{h_3}\right)^2 \end{bmatrix} X^T$$



# Metric

- Many methods are available in literature to construct the metric
- Most commonly used methods in FUN3D are based on a reconstructed Hessian (`adapt_hessian_method`) of a scalar (`adapt_hessian_key`), i.e. Mach number

## Metric Adaptation Mechanics

- Selectable with `adapt_library` in `&adapt_mechanics` or driven with scripts
- FUN3D is distributed with
  - refine/one (mature, development stopped)
  - refine/two (immature, under development, 2D, mixed elements)
- FUN3D can interact with external tools
  - BAMG (Bidimensional Anisotropic Mesh Generator)
  - Felflo.a (Loseille, INRIA)
- FUN3D has also been used with in-house proprietary tools by customers



## Venditti Adaptation Metric

- Default option of `adapt_error_estimation` in `&adapt_metric_construction`
- Output-based size specification scales the stretching and orientation of the Mach Hessian grid metric (Venditti and Darmofal)

$$M = \left| \frac{\partial^2 \mathrm{Mach}}{\partial x^2} \right| = X \begin{bmatrix} \left(\frac{1}{h_1}\right)^2 & & \\ & \left(\frac{1}{h_2}\right)^2 & \\ & & \left(\frac{1}{h_3}\right)^2 \end{bmatrix} X^T$$

## Venditti Adaptation Metric

- Output-based size specification scales the stretching and orientation of the Mach Hessian grid metric (Venditti and Darmofal)
- This error is typically evaluated on an embedded grid (with a large memory requirement) with an interpolated solution `adapt_error_estimation='embed'`
- `adapt_error_estimation='single'` is an single grid heuristic

$$e_\kappa = \frac{|(\hat{\lambda} - \bar{\lambda})R(\hat{u})| + |(\hat{u} - \bar{u})R_\lambda(\hat{\lambda})|}{2}$$

$$\frac{h_{\text{request}}}{h_{\text{current}}} = \left( \frac{e_{\text{tol}}}{\sum e_\kappa} \frac{e_{\text{tol}}}{N e_\kappa} \right)^\omega$$



## INRIA Optimal Goal-Based Metric

- Only implemented for Euler equations
- Adjoint gradient weighted Hessian of the flux
- No explicit dependence on the current grid
- `adapt_error_estimation='opt-goal'`
- See Loseille, Dervieux, and Alauzet JCP 2010 DOI:0.1016/j.jcp.2009.12.021 for details

# Feature Local-Error Metric

- Implemented in the Venditti framework where the nodal error estimate is replaced with a function of a solution scalar
  - `adapt_feature_scalar_key`
  - `adapt_feature_scalar_form`
- See Bibb, et al. AIAA-2006-3679 for details and Shenoy, Smith, Park AIAAJA 2014 DOI:10.2514/1.C032195 for a recent application

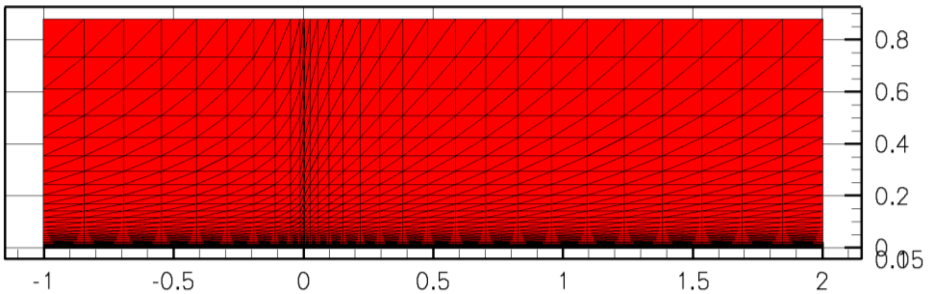


# Cases

- Single output-based cycle performed manually on a supersonic flat plate
- Semi-automatic feature-based adaptation to supersonic ramp
- Fully scripted diamond airfoil drag adaptation in supersonic flow

# Supersonic Flat Plate

- Mach 2, 1,000,000 Reynolds number, Spalart-Allmaras turbulence model






# Initial Flow Solution

- Initial `fun3d.nml` grid and flow conditions

```
&project
  project_rootname = "box01"
/
&raw_grid
  grid_format = "fast"
  data_format = 'ASCII'
/
&reference_physical_properties
  mach_number      = 2.0
  reynolds_number  = 1.0e+6
/
```

# Initial Flow Solution

- Initial `fun3d.nml` solver parameters

```
&nonlinear_solver_parameters
  schedule_iteration = 1    50
  schedule_cfl       = 1.0  200.0
  schedule_cflturb   = 1.0   10.0
/
&linear_solver_parameters
  linear_projection = .true.
  meanflow_sweeps   = 5
  turbulence_sweeps = 5
/
&code_run_control
  steps              = 1000
  stopping_tolerance = 1.0e-13
  restart_read       = "off"
/
```

Convergence of all residuals is critical!

Linear system Krylov projection



# Initial Flow Solution

- Initial `fun3d.nml` co-visualization

```
&global
  boundary_animation_freq = -1
/
&boundary_output_variables
  number_of_boundaries = -1 ! compute from list
  boundary_list        = '1-7'
  mu_t                 = .true.
/
```

# Initial Flow Solution

- Initial `fun3d.nml` co-visualization

```
&sampling_parameters
  number_of_geometries = 2
  sampling_frequency(1) = -1
    type_of_geometry(1) = 'plane'
      plane_center(:,1) = 0.0, 0.05, 0.0
      plane_normal(:,1) = 0.0, 1.0,  0.0
  sampling_frequency(2) = -1
    type_of_geometry(2) = 'plane'
      plane_center(:,2) = 1.0, 0.0, 0.0
      plane_normal(:,2) = 1.0, 0.0, 0.0
/
```



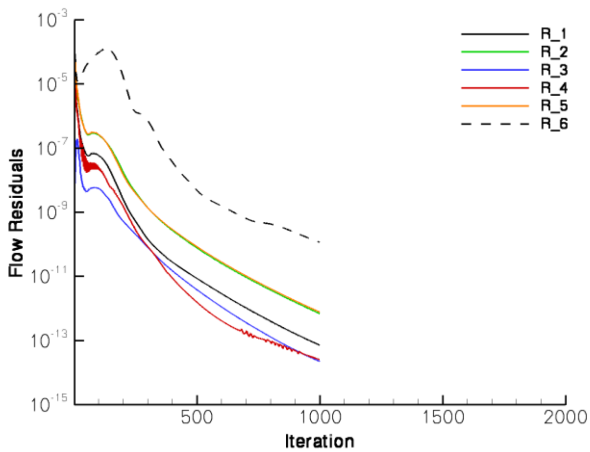
# Initial Flow Solution

- Follow the design directory layout convention
- Grid and `fun3d.nml` should be in a directory named `Flow`

```
$ cd Flow
$ mpirun -np 8 nodet_mpi
```

# Initial Flow Solution

- Flow solver (primal) convergence history






# Initial Adjoint Solution

- Adjoint function is defined in `rubber.data`
  - Only need to set the cost function, the other design inputs no used
- This is a integral of pressure along a line
  - Target off-body pressures required for sonic boom prediction

```
...
Components of func 1: boundary id (0=all)/name/value/weight/target/power
    0 boom_targ    0.000000000000000     1.0     0.00000 1.000
...
```

# Initial Adjoint Solution

- The `boom_targ` function requires an additional namelist in `fun3d.nml`

```
&sonic_boom
 x_lower_bound =  0.0
 x_upper_bound =  1.0
 nsignals = 1
 y_ray(1) =  0.05
 z_ray(1) =  0.1
/
```



# Initial Adjoint Solution

- Initial `fun3d.nml` adjoint solver parameters

```
&code_run_control
  steps              = 200
  stopping_tolerance = 1.0e-13
  restart_read       = "off"
/
```

Typically run less adjoint iterations

# Initial Adjoint Solution

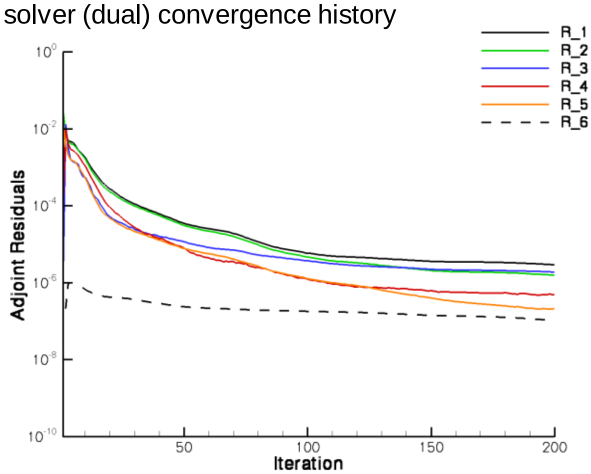
- Follow the design directory layout convention
- Grid and `fun3d.nml` should be in a directory named `Flow`
- The file `rubber.data` should be in the directory above
- Adjoint solver should be run in a directory named `Adjoint`

```
$ cd Adjoint
$ mpirun -np 8 dual_mpi --outer_loop_krylov
```



# Initial Adjoint Solution

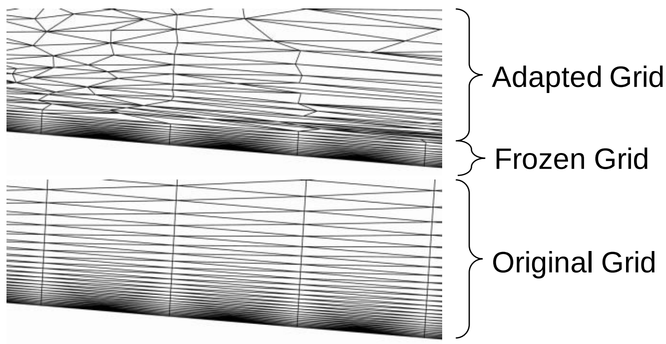
- Adjoint solver (dual) convergence history



# Output-Based Adaptation

- Output-based adaptation `fun3d.nml` parameters

```
&adapt_mechanics
  adapt_project   = 'box02'      New project name
  adapt_freezebl = 0.001         Frozen boundary layer
/
```






# Output-Based Adaptation

- Planar geometry is specified to `refine/one` with `faux_geom`
- Place in the same directory that the adaptation is executed (`Adjoint`)

```
7                                         Number of planes
1 xplane      -1.0000000000000000
2 xplane       2.0000000000000000
3 yplane       0.0000000000000000
4 yplane       0.1000000000000000         Each plane with normal
5 zplane       0.0000000000000000         and position
6 zplane       0.0000000000000000
7 zplane       0.8813629407814508
```

# Initial Adjoint Solution

- Follow the design directory layout convention
- Grid and `fun3d.nml` should be in a directory named `Flow`
- The file `rubber.data` should be in the directory above
- Adjoint grid adaptation should be run in a directory named `Adjoint`

```
$ cd Adjoint
$ mpirun -np 8 dual_mpi --rad --adapt
```

`--rad` = Residual Adjoint Dot-product

`--adapt` = Activates grid adaptation



# Adapted Flow Solution

- Initial `fun3d.nml` grid and flow conditions

```
&project
  project_rootname = "box02"
/
&raw_grid
  grid_format = "aflr3"
  data_format = 'stream'
/
&code_run_control
  steps              = 1000
  stopping_tolerance = 1.0e-13
  restart_read       = "on"
/
```

New project name

New grids are always AFLR3 (ugrid) stream format

The solution is interpolated

# Adapted Flow Solution

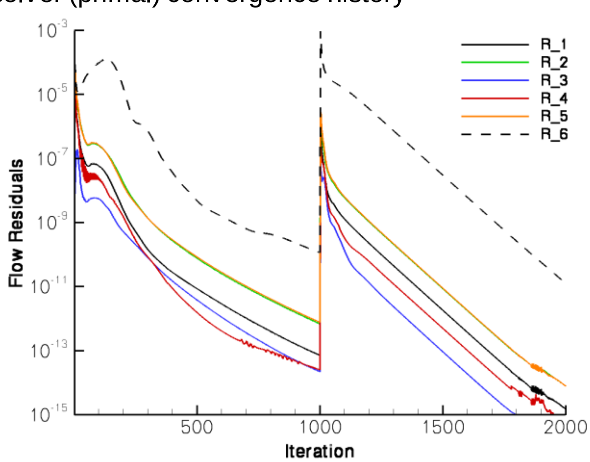
- Follow the design directory layout convention
- Grid and `fun3d.nml` should be in a directory named `Flow`

```
$ cd Flow
$ mpirun -np 8 nodet_mpi
```

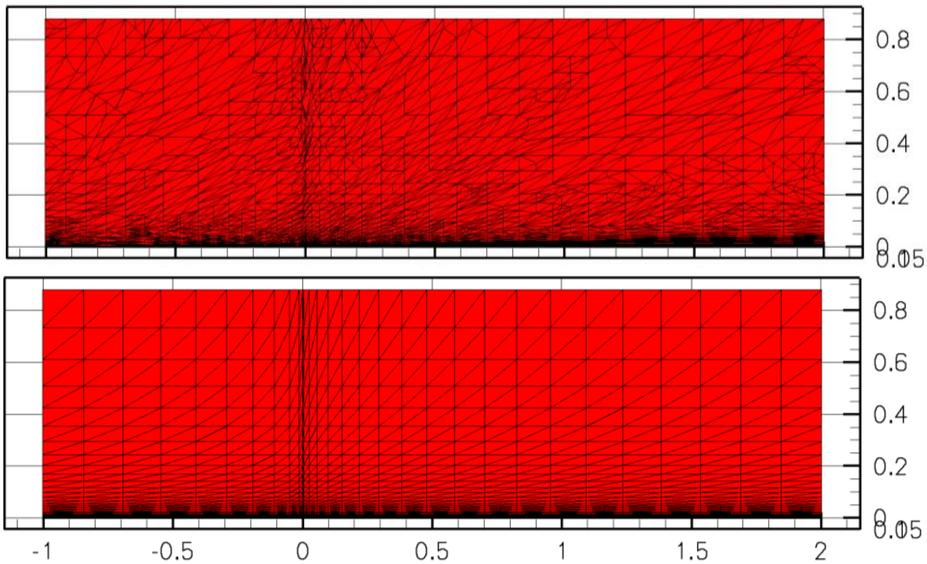


# Adapted Flow Solution

- Flow solver (primal) convergence history



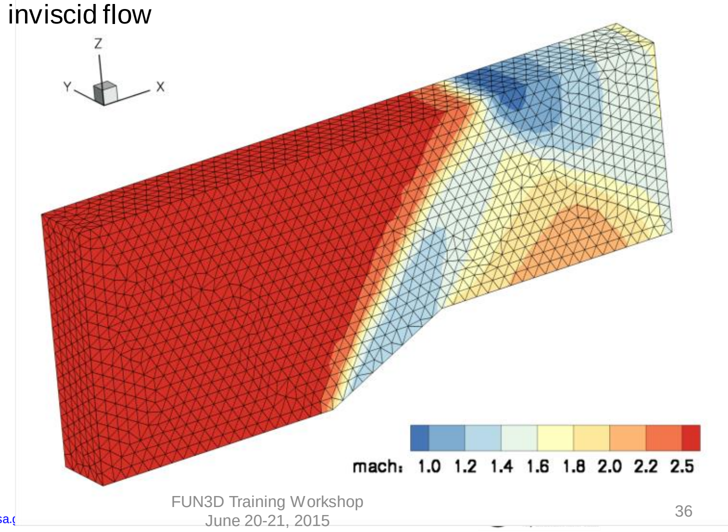
## Adapted and Original Flat Plate Grid






## Supersonic Ramp Feature Adaptation

- Mach 2.5, inviscid flow



# Initial Flow Solution

- Initial `fun3d.nml` grid and flow conditions

```
&project
  project_rootname = 'ramp00'
/
&raw_grid
  grid_format = 'aflr3'
  data_format = 'stream'
/
&governing_equations
  viscous_terms = 'inviscid'
/
&reference_physical_properties
  mach_number = 2.5
/
```



# Initial Flow Solution

- Initial `fun3d.nml` grid and flow conditions

```
&inviscid_flux_method
  first_order_iterations = 10000
  flux_construction      = 'vanleer'
/
&nonlinear_solver_parameters
  schedule_iteration =    1    20
  schedule_cfl       = 10.0 1000.0
/
```

First-order and large CFL for demonstration, use second order with frozen limiter in practice

# Initial Flow Solution

- Grid and `fun3d.nml` should be in the current directory

```
$ cd Flow
$ mpirun -np 8 nodet_mpi --irest 0
```

`--rest 0` turns off restart for the initial grid



# Feature-Based Adaptation

- Output-based adaptation `fun3d.nml` parameters

```
&adapt_mechanics
  adapt_project = 'ramp01'
  adapt_cycles  = 10
/
&adapt_metric_construction
  adapt_feature_scalar_key  = 'mach'
  adapt_feature_scalar_form = 'delta-l'
  adapt_output_tolerance    = 0.05
  adapt_min_edge_length     = 0.01
  adapt_max_anisotropy      = 1.0
/
```

New project name

Passes for grid mechanics

Target shocks and expansions

Isotropic

# Feature-Based Adaptation

- Planar geometry is specified to `refine/one` with `faux_geom`
- Place in the same directory that the adaptation is executed

```
8
1  zplane   0.0
2  zplane   2.0
3  yplane   0.0
4  xplane  -2.0
5  yplane   0.5
6  xplane   3.0
7  general_plane 0.0
   -0.5 0.0 1.0
8  zplane   0.5
```

Number of planes

Each plane with normal and position



# Feature-Based Adaptation

- Grid, `fun3d.nml`, and restart should be in the current directory

```
$ mpirun -np 8 nodet_mpi --adapt
```

`--adapt` use adaptation mechanics

# Scripting it

- Unix `bash` and `sed` are your friends
- Create `fun3d.nml` files ahead of time

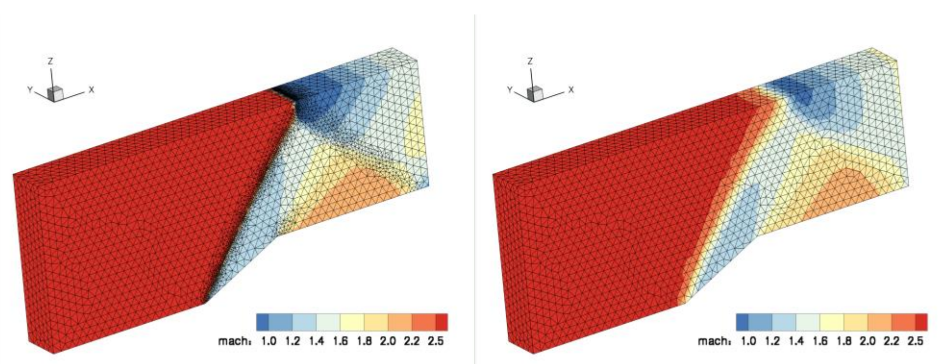
```
#! /bin/bash

cp -f fun3d.nml-00 fun3d.nml
mpirun -np 8 nodet_mpi --irest 0

for i in {1..5} ; do
  mpirun -np 8 nodet_mpi --adapt
  cp -f fun3d.nml-0${i} fun3d.nml
  mpirun -np 8 nodet_mpi
done
```

Initial solution

Adapt and flow solve on new grid



# Adapted and Original Grid and Mach Number



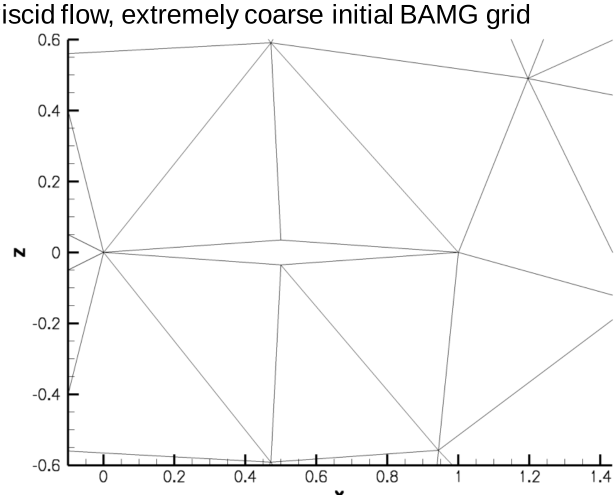
## F3D script

- Domain specific language written in Ruby
- Simple syntax for driving adaptation with the power of a scripting language if needed
- Input file `case_specifics` is scanned for updates during adaptation allowing for computational steering
- All input files are expected to be in the current directory and are also scanned for updates
  - Files are copied to `Flow` and `Adjoint` as needed
- Can generate `rubber.data` with `$ f3d function cd`
- Subcommands to start, stop, and examine adaptation in progress
- Discussed in Grid Adaptation section of the user manual



## Drag-Adapted Diamond Airfoil

- Mach 2.0, inviscid flow, extremely coarse initial BAMG grid



# `F3D` input `case_specifics` example

- Keyword value pairs to add command line options, adjust namelist settings, and specify outer adaptation cycle iterations

```
root_project 'diamond'

number_of_processors 8

adj_cl " --outer_loop_krylov "

rad_nl["adapt_complexity"] = 200*(1.5**iteration)

all_nl['data_format']="'stream'" if (iteration>1)

first_iteration 1
last_iteration 10
```



# Namelist Setup

- Initial `fun3d.nml` grid and flow conditions

```
&project
  project_rootname = 'diamond01'
/
&raw_grid
  grid_format = 'aflr3'
  data_format = 'ascii'
/
&code_run_control
  steps             = 500
  stopping_tolerance = 1.0e-11
  restart_read       = 'off'
/
```

# Namelist Setup

- Initial `fun3d.nml` grid and flow conditions

```
&inviscid_flux_method
  kappa_umuscl = 0
  flux_limiter = 'hvanalbada'
  freeze_limiter_iteration = 100
  flux_construction      = 'vanleer'
/
```



# Namelist Setup

- Initial `fun3d.nml` grid and flow conditions

```
&adapt_mechanics
 adapt_library = 'refine/two'
 adapt_project = 'diamond02'
/
&adapt_metric_construction
  adapt_hessian_method = 'grad'
  adapt_hessian_average_on_bound = .true.
  adapt_twod = .true.
  adapt_statistics = 'average'
  adapt_max_anisotropy = 10.0
  adapt_complexity = 1000
  adapt_gradation  = 1.5
  adapt_current_h_method = 'implied'
/
```

refine version 2 mechanics

# F3D script

- Run with no subcommands for help

```
$ f3d
usage: f3d <command>

 <command>       description
 ---------       -----------
 start           Start adaptation
 view            Echo a single snapshot of stdout
 watch           Watch the result of view
 shutdown        Kill all running fun3d and ruby processes
 clean           Remove output and sub directories
 function [name] write rubber.data with cost function [name]
```



# F3D script

- To begin and watch progress

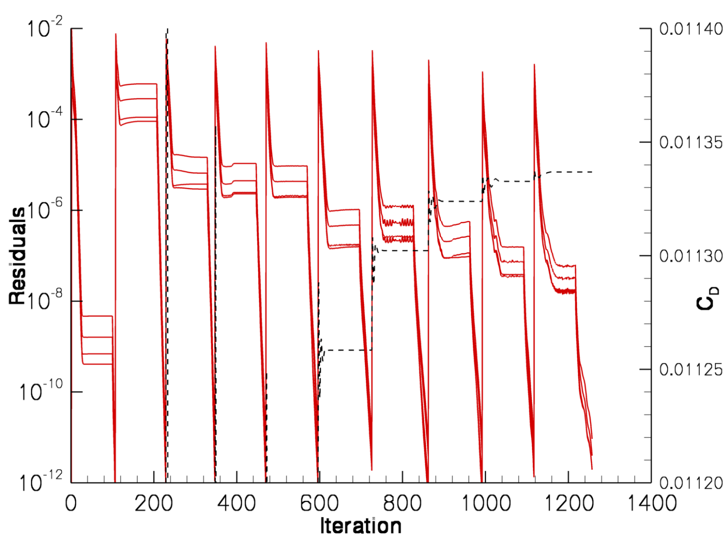
```
$ f3d start
$ f3d watch
```

# F3D script

- Copies `fun3d.nml` into Flow directory and modifies it to set `project_rootname`, `restart_read`, and other options with the `nl_flo`, `nl_adj`, `nl_rad` hashes
- Backup copies of `fun3d.nml` are saved as `[project]_flow_fun3d.nml`, `[project]_dual_fun3d.nml`, and `[project]_rad_fun3d.nml`
- Backup copies of standard screen output are are saved as `[project]_flow_out`, `[project]_dual_out`, and `[project]_rad_out`

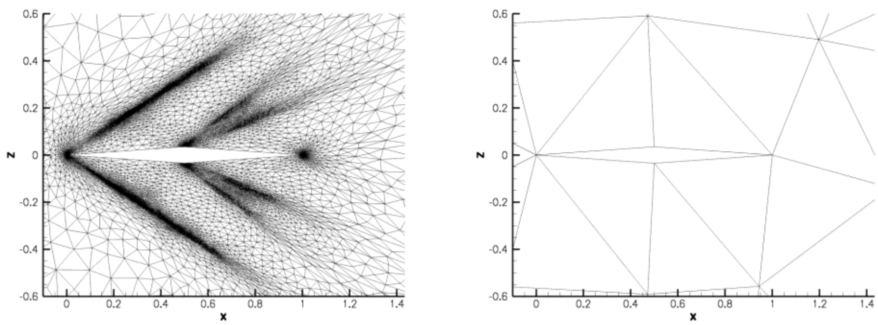


# Drag-Adapted Diamond Airfoil



# Drag-Adapted Diamond Airfoil

- Mach 2.0, inviscid flow






# Running with PBS

- Creates and submits a PBS batch script

```
$ pbswrap
Usage: pbswrap [OPTION]... [COMMAND]
 required:
  -cpn P  there are P cores per node
  -t   H  walltime limit of H hours
  -np  C  run on C cores (-np and -n are exclusive)
  -n   N  run on N nodes (-np and -n are exclusive)
 optional:
  -q   Q  submit to queue Q otherwise try system default
  -a   A  charge job to account A
  -m   M  use cpu model M
  -b      block the pbs submission
```

# Running with PBS

- Example usage in `case_specifics`
- Generates a pbs job with xMoDaHrMnS.pbs
  - Month, Day, Hour, Min, tens of Sec.
- Output written to file named xMoDaHrMnS

```
mpirun_command 'pbswrap -b -q K3-standard -cpn 16 -t 1'
```



# What Can Go Wrong?

- Flow solver did not produce a project.forces file on completion
  - Indicate a setup problem (first iteration)
  - Previous grid adaptation failed (error estimation, grid mechanics)
  - Flow solver crashed or diverged
- Examine `flow_out` for more details

```
/u/mpark/fun3d/opt/bin/f3d:149:in `readlines': No such file or
directory - Flow/diamond07.forces (Errno::ENOENT)
    from /u/mpark/fun3d/opt/bin/f3d:149:in `read_forces'
    from /u/mpark/fun3d/opt/bin/f3d:121:in `flo'
    from /u/mpark/fun3d/opt/bin/f3d:224:in `iteration_steps'
    from /u/mpark/fun3d/opt/bin/f3d:233:in `iterate'
    from /u/mpark/fun3d/opt/bin/f3d:310
```

## What Can Go Wrong?

- Adjoint solver setup (particularly `rubber.data`)



## Evolving Process

- AVIATION paper for status
  - Session: CFD-03, Meshing Techniques I, Monday, June 22, 2015 from 9:00 AM to 12:30
- Things learned will be shift to default options
- Continuing development of refine grid mechanics
- Implementation of error estimation technics