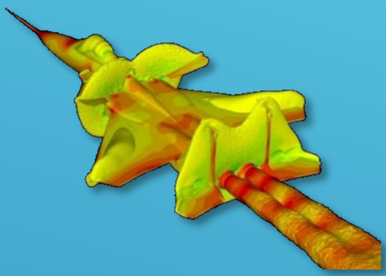
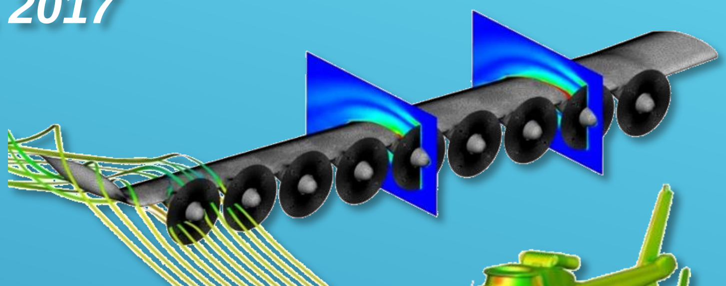
Fully Unstructured Navier-Stokes

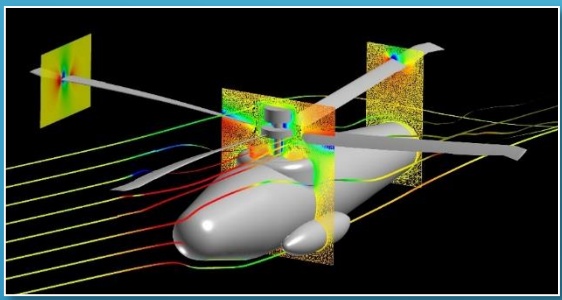
Training Workshop

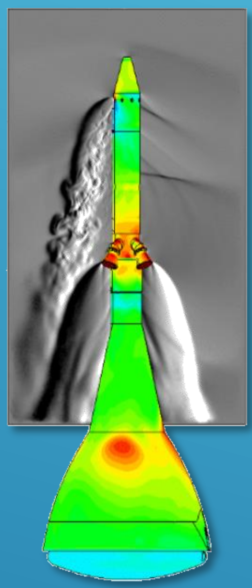
Waikoloa, Hawaii

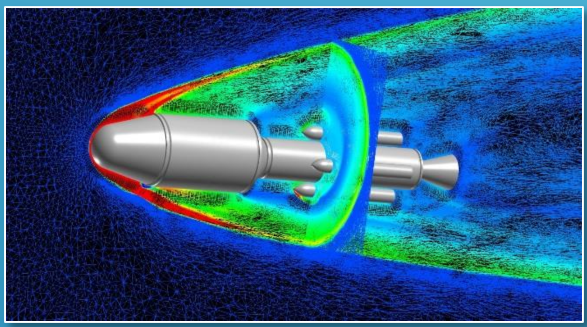
July 30, 2017

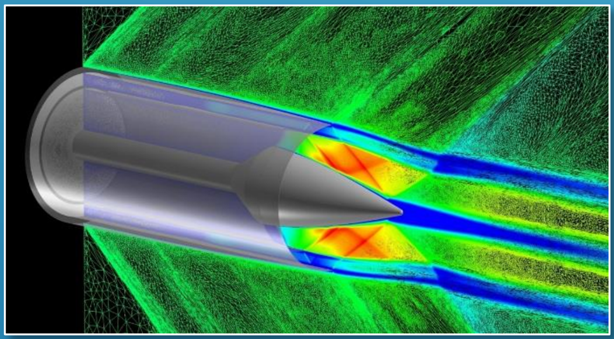


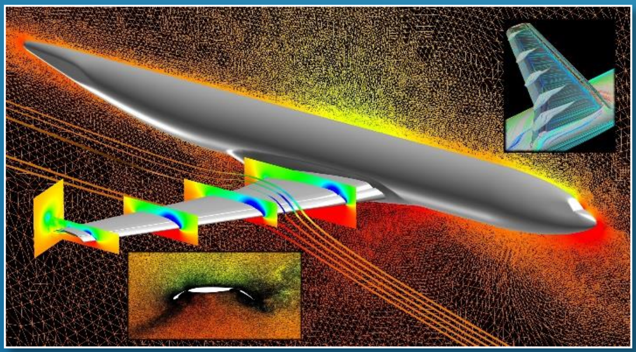


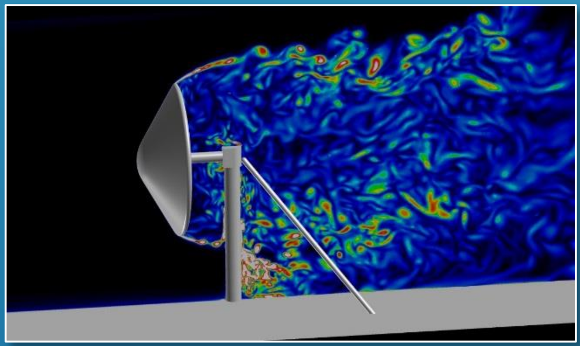


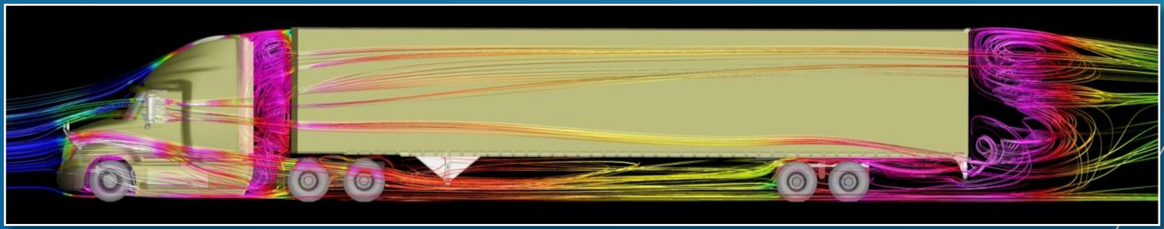












Some images courtesy Ashley Korzun, BMI Corporation, Chris Heath, Karen Deere, Mark Moore, Sally Viken, and US Army

FUN3D Training Evaluation Form

I am a CFD:	❍ Novice	❍ Experienced user	❍ Expert
I am a FUN3D:	❍ Novice	❍ Experienced user	❍ Expert

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1. The training met my expectations.	❍	❍	❍	❍	❍
2. I will be able to apply the knowledge learned.	❍	❍	❍	❍	❍
3. The training objectives for each topic were identified and followed.	❍	❍	❍	❍	❍
4. The content was organized and easy to follow.	❍	❍	❍	❍	❍
5. The materials distributed were pertinent and useful.	❍	❍	❍	❍	❍
6. The trainers were knowledgeable.	❍	❍	❍	❍	❍
7. The quality of instruction was good.	❍	❍	❍	❍	❍
8. The trainers met the training objectives.	❍	❍	❍	❍	❍
9. Class participation and interaction were encouraged.	❍	❍	❍	❍	❍
10. Adequate time was provided for questions and discussion.	❍	❍	❍	❍	❍

11. How do you rate the training overall?

Excellent	Good	Average	Poor	Very poor
❍	❍	❍	❍	❍

12. What aspects of the training could be improved?

13. Other comments?

Your name (optional):

THANK YOU FOR YOUR PARTICIPATION!

FUN3D v13.1 Training

Session 1:
Meet and Greet

Who is Here From the NASA Team?

From NASA Langley in Hampton, Virginia:

Eric Nielsen, Computational AeroSciences Branch
- 25 years with the FUN3D effort
- Adjoint-based sensitivity analysis and design
- High-performance computing
- General all-around code monkey

Many other FUN3D team members at LaRC and other sites, with very diverse expertise, interests, and backgrounds

Who Are You?

Name

Organization/Company and Location

Any CFD Experience?

Any FUN3D Experience?

Goals/Uses for FUN3D

FUN3D v13.1 Training

Session 2:
Welcome and Overview

Eric Nielsen

FUN3D Training Workshop
July 30, 2017

Session 1: Introductions	8:00-8:15
Session 2 Welcome and Overview	8:15-8:45
Session 3: Compilation and Installation	8:45-9:00
Session 4: Gridding, Solution, and Visualization Basics	9:00-10:30
BREAK	10:30-10:45
Session 5: Adjoint-Based Design for Steady Flows	10:45-12:00

Administrative Details

- Need to stay on schedule, but please do not hesitate to ask questions
- In-room wireless access:
  - Network: Password:
- Please submit your evaluation form at the end of the workshop
  - Very interested in your feedback, good or bad!

All Material Available Online

- For the v13.1 material presented here:
  - Slides online in PDF format
  - To obtain FUN3D, see website for link to NASA Software Catalog
- A FUN3D v13.1 manual is available as NASA/TM-2017-219580 on the website
  - You should also receive a copy of this with the source code distribution
  - Additional material will continue to be added with new releases
  - Your feedback/suggestions are extremely helpful
- Extensive material from prior training workshops is available on the website
  - Slides in PDF
  - Pro-shot streaming video
  - Demo content can be downloaded as a tarball
- We hope to eventually add an extensive tutorials document

The FUN3D Development Team

fun3d-developers@lists.nasa.gov

- Consists of ~15-20 researchers across several branches at Langley
  - Computational AeroSciences Branch
  - Aerothermodynamics Branch
- Some people are full-time FUN3D, others part-time
  - Spectrum runs from full-time development to full-time applications
- Also external groups such as Georgia Tech, National Institute of Aerospace (NIA)
- Open to other interested parties joining us
  - Remote, real-time, read/write access to FUN3D repository is available

The FUN3D Support Team

fun3d-support@lists.nasa.gov

"Who sees my questions to the support alias?"

- Consists of 16 members of the development team
- All are NASA civil servants
  - Proprietary/sensitive data can be shared/discussed: all are bound by Trade Secrets Act
- Members: Kyle Anderson, Bob Biedron, Jan-Renee Carlson, Cameron Druyor, Peter Gnoffo, Dana Hammond, Bill Jones, Bil Kleb, Beth Lee-Rausch, Steve Massey, Eric Nielsen, Matt O'Connell, Mike Park, Kyle Thompson, Aaron Walden, Jeff White

Myth: Our job is to develop a production-level tool and support users.

Reality: ***None*** of us are funded at ***any*** level to support users, maintain documentation, keep up a website, run training workshops, etc. The team is funded solely to perform their individual research efforts.

The FUN3D User Community

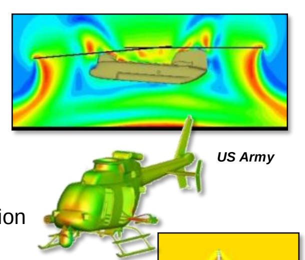
fun3d-users@lists.nasa.gov

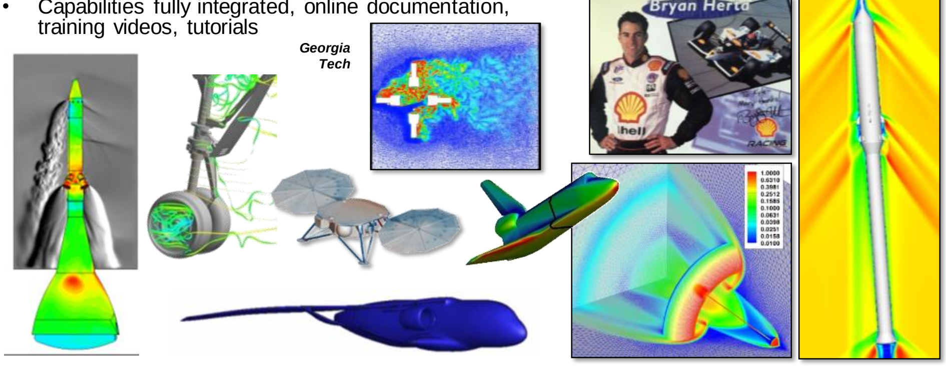
- FUN3D widely used within NASA for projects across the speed range
  - Both engineering and research applications
  - Users routinely running on several thousand cores
- Distributed to hundreds of external organizations across academia, industry, DoD, and OGAs
  - Average about 150 distributions / year
  - Wide range of uses including aerospace, automotive, HPC, wind energy, etc.
  - Wide range of hardware being used
  - From RC enthusiasts on single workstation to groups generating matrices of hundreds of solutions on thousands of HPC nodes

http://fun3d.larc.nasa.gov FUN3D Training Workshop July 30, 2017 7

FUN3D Core Capabilities

- Established as a research code in late 1980s; now supports numerous internal and external efforts across the speed range
- Solves 2D/3D steady and unsteady Euler and RANS equations on node-based mixed element grids for compressible and incompressible flows
- General dynamic mesh capability: any combination of rigid / overset / morphing grids, including 6-DOF effects
- Aeroelastic modeling using mode shapes, full FEM, CC, etc.
- Constrained / multipoint adjoint-based design and mesh adaptation
- Distributed development team using agile/extreme software practices including 24/7 regression, performance testing
- Capabilities fully integrated, online documentation, training videos, tutorials

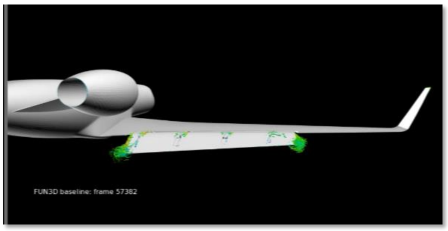


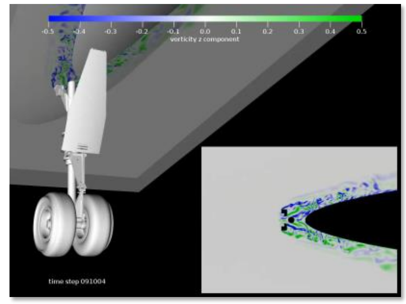


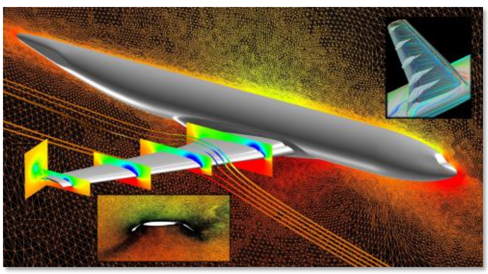
Some Recent NASA Applications

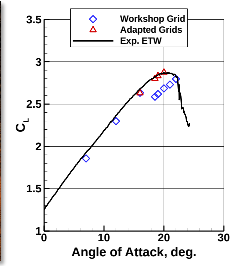
Airframe Noise

Courtesy NASA/Gulfstream Partnership on Airframe Noise Research





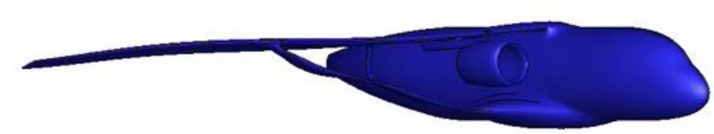




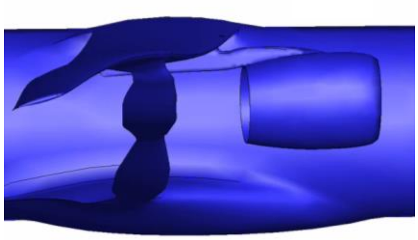
Adjoint-Based Adaptation for High-Lift

10

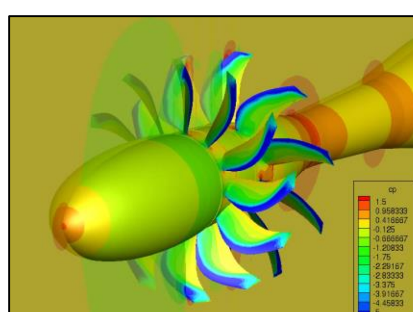
Some Recent NASA Applications



Courtesy Bob Bartels



Open-Rotor Concepts



Aeroelastic Analysis of the Boeing SUGAR Truss-Braced Wing Concept

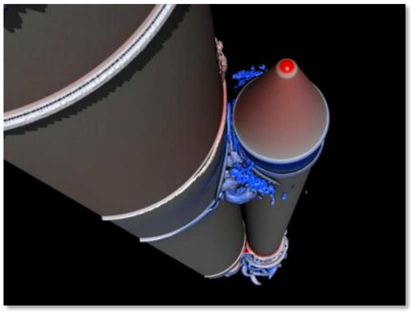
Courtesy Bill Jones

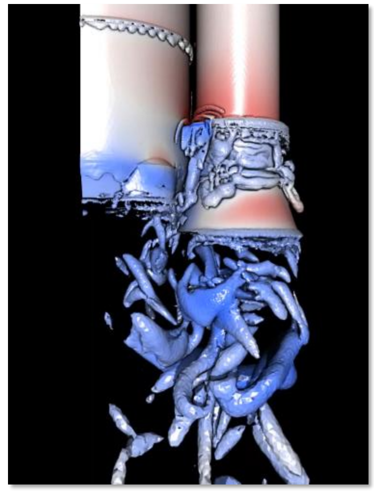
http://fun3d.larc.nasa.gov

FUN3D Training Workshop
July 30, 2017

11

Some Recent NASA Applications



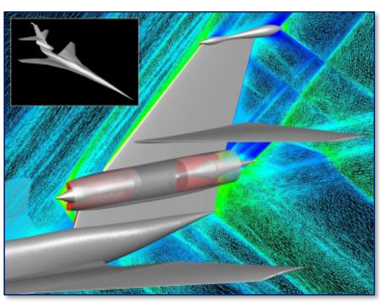


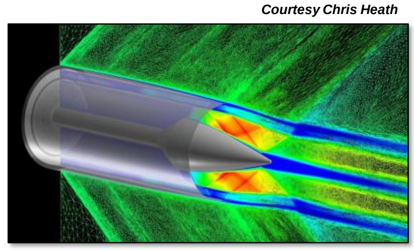
Transonic Buffet Characterization for Space Launch System

Courtesy
Greg Brauckmann,
Steve Alter, Bil Kleb

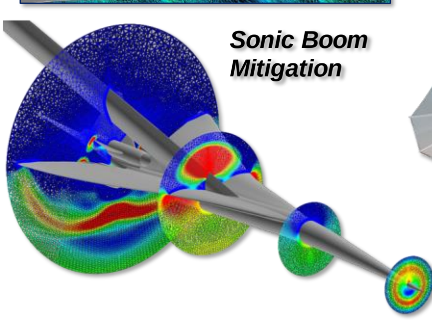
Some Recent NASA Applications

Courtesy Chris Heath





Sonic Boom Mitigation

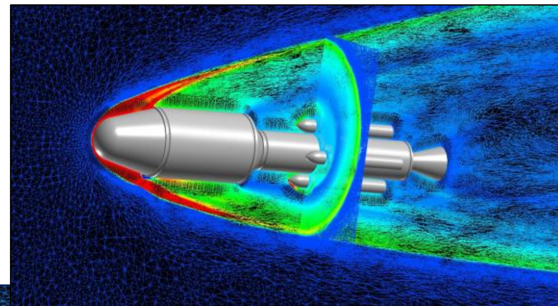


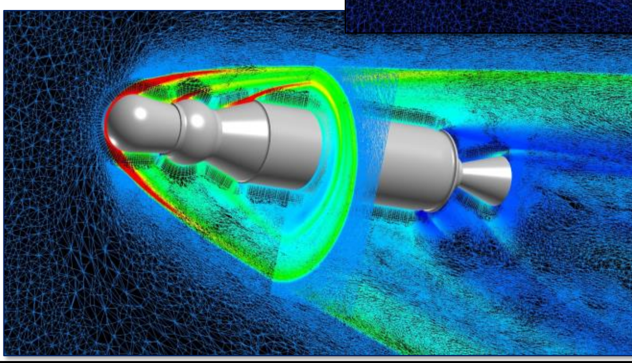
Mars InSight Lander

Some Recent NASA Applications

Mars Ascent Vehicle for Sample Return

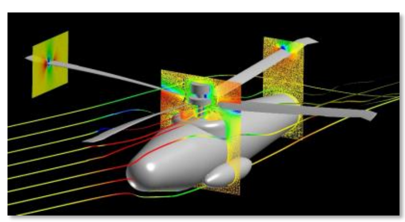
Courtesy Ashley Korzun





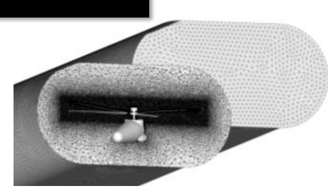
Some Recent NASA Applications

Validation for Full Scale UH60A

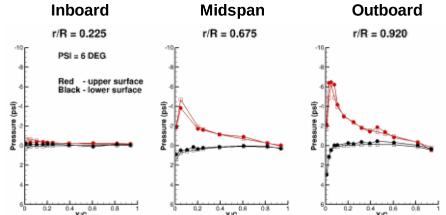


Courtesy Beth Lee-Rausch, Bob Biedron

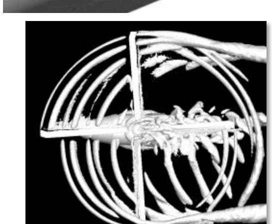
- Structural loads
- Sectional airloads/pressures
- Balance loads
- Control settings
- Blade root motions
- Elastic blade deflections



Inboard **Midspan** **Outboard**

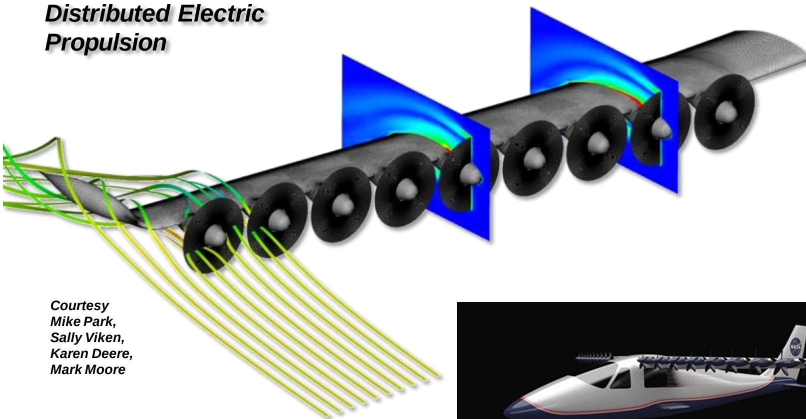


Blade Pressures at High Advance Ratio



Some Recent NASA Applications

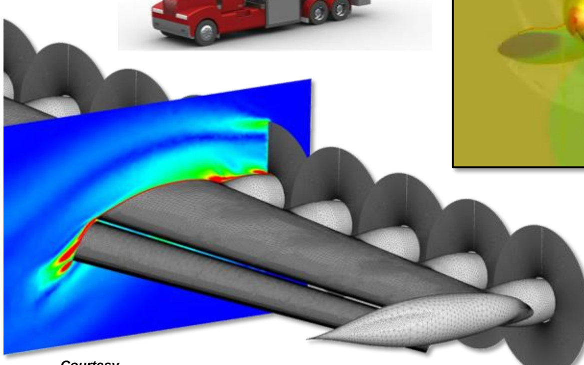
Distributed Electric Propulsion

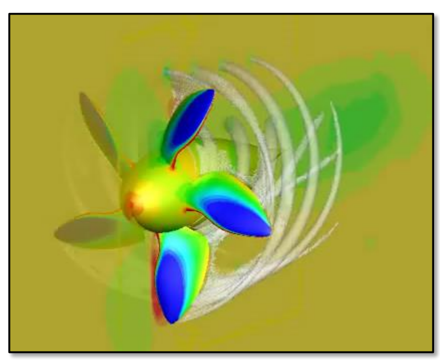


Courtesy
Mike Park,
Sally Viken,
Karen Deere,
Mark Moore

http://fun3d.larc.nasa.gov FUN3D Training Workshop July 30, 2017 FUN3D 16

Some Recent NASA Applications





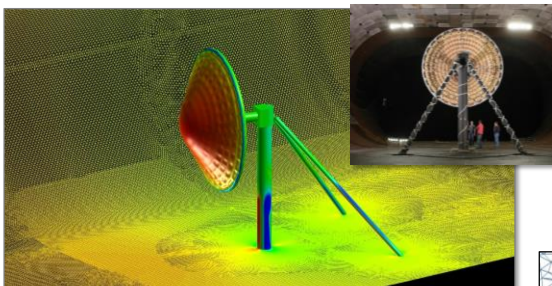
Courtesy Bill Jones

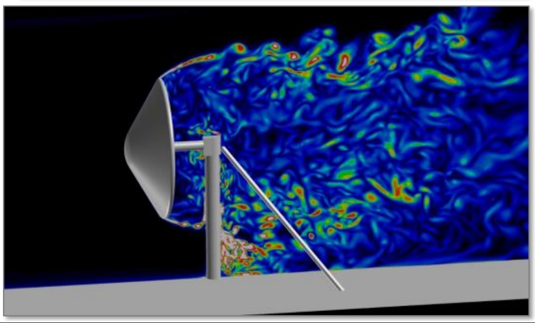
Distributed Electric Propulsion

Courtesy
Mike Park, Sally Viken,
Karen Deere, Mark Moore

http://fun3d.larc.nasa.gov FUN3D Training Workshop July 30, 2017 FUN3D 17

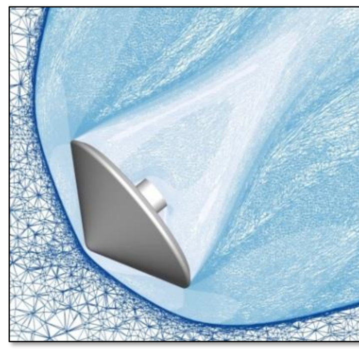
Some Recent NASA Applications





Aeroelastic Analysis of HIADs: Hypersonic Inflatable Aerodynamic Decelerators

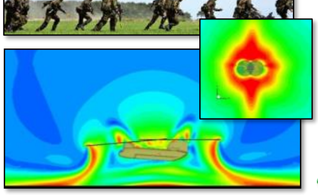
Courtesy Beth Lee-Rausch, Bob Biedron, and Bil Kleb

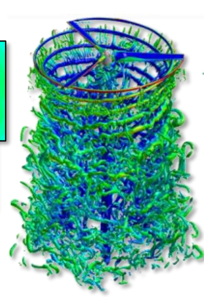


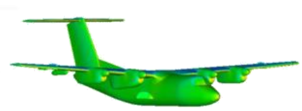
At the Department of Defense

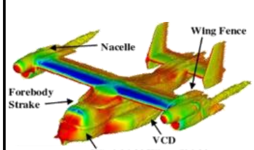
AMRDEC at Redstone Arsenal

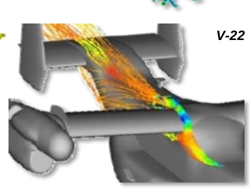
- **Troop safety:** airworthiness qualification
- **Dramatic cost savings:** fewer tunnel & flight tests
- Intense demand for timely results on massive computing systems
- Decade of use in direct support of the US warfighter











- **NAVAIR at Patuxent River**
- **Air Force Research Laboratory**
- **HPCMP CREATE-AV**

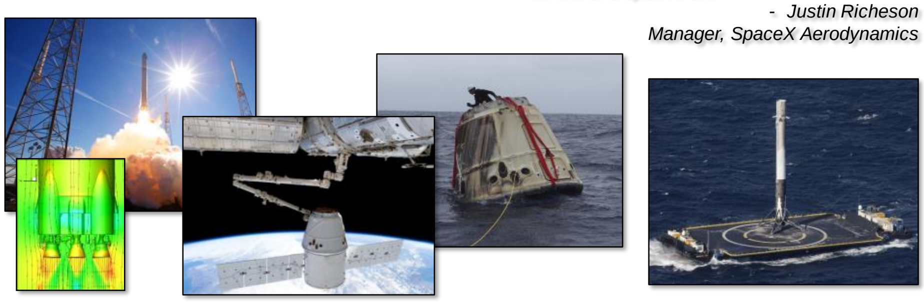
Across the Aerospace Industry

First private company to achieve orbit and dock with the International Space Station

- FUN3D used for extensive analysis of Falcon 1 and Falcon 9 rockets, Dragon spacecraft
- Team consults frequently and provides new features and capabilities as requested

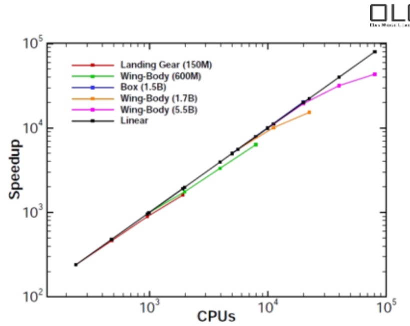
"The FUN3D software suite and development team have enabled SpaceX to rapidly design, build, and successfully fly a new generation of rockets and spacecraft."

- Justin Richeson
Manager, SpaceX Aerodynamics



FUN3D and High-Performance Computing

FUN3D is used on a broad range of HPC installations around the country



Scaled to 80,000 cores on DoE's Cray XK7 'Titan' using grids containing billions of elements

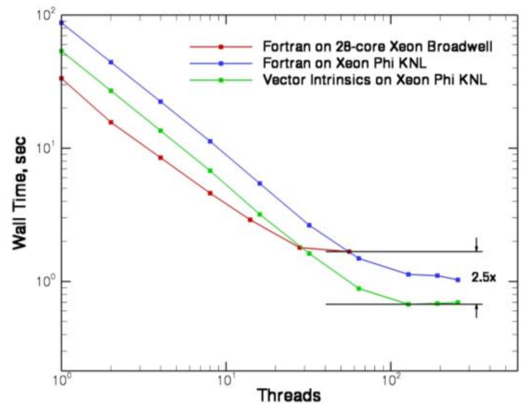
Awarded the Gordon Bell Prize in a past collaboration with Argonne National Lab

http://fun3d.larc.nasa.gov

FUN3D Training Workshop
July 30, 2017

20

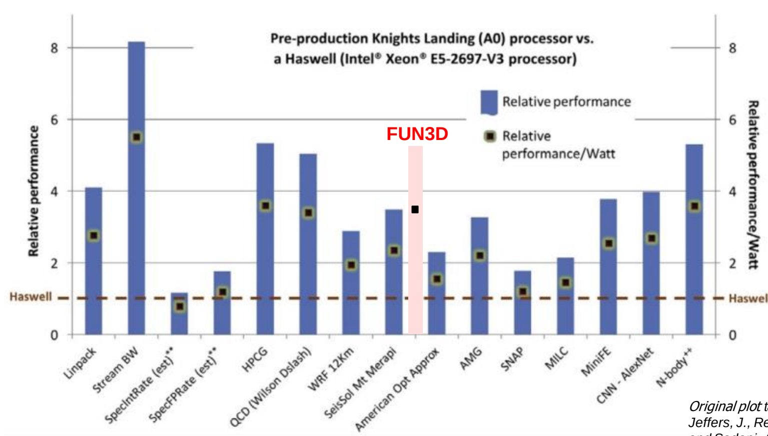
FUN3D Solver on Intel Xeon Phi



- "Works out of the box" paradigm for KNL is encouraging but dangerous: tempting to declare success before achieving its full potential
- Vector intrinsics on Xeon Phi Knights Landing beat conventional Fortran on 28-core Xeon Broadwell by 2.5x
- Intrinsics attractive for performance (including Skylake and beyond), but effort/portability must be considered

http://fun3d.larc.nasa.gov
FUN3D Training Workshop
July 30, 2017
21

FUN3D Solver on Intel Xeon Phi

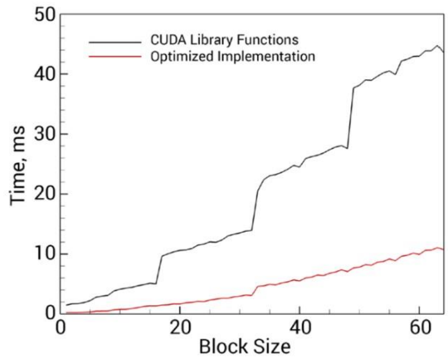


Original plot taken from Jeffers, J., Reinders, J., and Sodani, A., "Intel Xeon Phi High Performance Programming, Knights Landing Edition," 2016.

- Using the more common single-socket Haswell benchmark, Knights Landing is 5.4x faster and 3.6x more power-efficient
- Compares well with other early apps

http://fun3d.larc.nasa.gov
FUN3D Training Workshop
July 30, 2017
22

FUN3D Solver on NVIDIA GPUs



- FUN3D implicit solver also implemented for GPUs using OpenACC, CUDA, and PTX
- Up to 7x improvement over existing CUDA libraries for range of block sizes
- NVIDIA Pascal P100 shows 3.9x speedup over 28-core Xeon Broadwell

Some Final Notes

- The material that will be shown here represents the current recommended best practices for the perfect gas option in FUN3D
- Many topics omitted from what is normally a two-day course:
  - Boundary conditions, turbulence models, high-speed simulations, geometry parameterization, error estimation and mesh adaptation, time-dependent flows, dynamic and overset grid simulations, adjoints for unsteady flows, aeroelastic simulations, rotorcraft simulations, general-gas simulations
- There are always many research and development efforts taking place within the code that are not described here
- If you do not see something, please ask about it

FUN3D v13.1 Training Session 3: Compilation and Installation

Eric Nielsen

Learning Goals

- What this will teach you
  - How to configure and compile the FUN3D suite
  - Configuration options
    - Enable/Disable capabilities
    - Specify the location of 3rd party libraries and tools
  - How we do it
- What you will not learn
  - How to build/install 3rd party libraries and tools
  - How to configure your system to compile Fortran 90/MPI code
- What should you already know
  - How to navigate through a *NIX shell
    - **`mkdir`**
    - **`cd`**
    - Absolute/relative paths

Setting

- Background
  - FUN3D uses the de facto industry standard build environment provided by GNU Autotools
  - Build of the FUN3D distribution does **not** require Autotools on your system
  - Provides localization through options to a configuration script
- Compatibility
  - Requires a Bourne Shell derivative (*NIX, OS X, MinGW, etc.)
  - Requires GNU \``make`\`
  - Requires a functioning Fortran 95 compliant compiler (some optional capabilities rely on Fortran 2003 additions)
  - May not work with non-standard installation of 3rd party libraries
  - DiRTLib and SUGGAR++ assumptions for overset support
  - Required library names: `libp3d.a, libdirt.a, libdirt_mpich.a, libsuggar.a, and libsuggar_mpi.a`
  - Developers will need GNU Autotools installed

Nuts and Bolts (1 of 4)

- Two step process
  - \``configure`\` selects capabilities and localizes to system
  - \``make`\` creates executables
- Distribution contains a \``configure`\` script
  - Familiar to Linux users/administrators who have built open source packages
  - Must <u>**NOT**</u> be edited by hand
  - Custom localization through command line options
- The \``configure`\` script creates **Makefiles**
  - **Makefiles** are customized/localized for a specific configuration
  - Not practical for human consumption
  - Must <u>**NOT**</u> be edited by hand
  - All localization is managed through the \``configure`\` script
  - Checks various details required by compilation
  - <u>Fails</u> <u>fast</u> (prior to compilation of FUN3D) if problems are detected with the configuration options (no compiler, missing libraries, etc.)

Nuts and Bolts (2 of 4)

- `configure --help` will show a list of all options
  - Command line options
  - Environment variables
  - Order independent (uses last value if specified multiple times)
- FUN3D optional Features of general interest

`--disable-FEATURE`	do not include FEATURE (same as `--enable-FEATURE=no`)
`--enable-FEATURE[=ARG]`	include FEATURE `[ARG=yes]`
`--enable-hefss`	build with High Energy Physics `[no]`
`--enable-ftune`	tailor Fortran compiler options for FUN3D `[yes]`

Nuts and Bolts (3 of 4)

- FUN3D optional Packages of general interest

```
--with-PACKAGE[=ARG]        use PACKAGE [ARG=yes]
--without-PACKAGE           do not use PACKAGE (same as --with-PACKAGE=no)
```

`--with-mpi[=ARG]`	Path to MPI library	(installation root)
`--with-mpibin[=ARG]`	MPI binary directory	(relative, absolute, without)
`--with-mpif90[=ARG]`	MPI Fortran compiler wrapper	(relative, absolute, without)
`--with-mpicc[=ARG]`	MPI C compiler wrapper	(relative, absolute, without)
`--with-mpiexec[=ARG]`	MPI execution startup script	(relative, absolute, without)
`--with-mpibin[=ARG]`	MPI bin directory	(relative, absolute, without)
`--with-mpiinc[=ARG]`	Path to "mpif.h"	(relative, absolute, without)
`--with-parmetis[=ARG]`	ParMetis install path	(contains lib/libparmetis.a)
`--with-dirtlib[=ARG]`	use DiRTlib overset library	(contains lib/libdirt.a)
`--with-suggar[=ARG]`	use SUGGAR overset library	(contains lib/libsuggar.a)
`--with-tecio[=ARG]`	Tecplot I/O library install path	(contains lib/libtecio.a)
`--with-refine[=ARG]`	use refine adaptation package	(installation root)
`--with-refineFAKEGeom[=ARG]`	to specify refine FAKEGeom libs `[-lFAUXGeom]`	
`--with-knife[=ARG]`	use Knife cut cell package	(installation root)
`--with-CGNS[=ARG]`	CGNS library path	(installation root)
`--with-PORT[=ARG]`	use PORT optimization library	(contains lib/libport.a)
`--with-KSOPT[=ARG]`	use KSOPT optimization library	(contains lib/libksopt.a)
`--with-SNOPT[=ARG]`	use SNOPT optimization library	(contains lib/libsnopt.a)

Nuts and Bolts (4 of 4)

- FUN3D environment variables of general interest

  FC — **`Fortran compiler command`**
  (overridden by `--with-mpif90`)

  FCFLAGS — **`Fortran compiler flags`**
  (adds to default unless --disable-ftune)

  LDFLAGS — **`linker flags, e.g. -L<libdir>`**
  if you have libraries in a nonstandard directory **`<libdir>`**

  CC — **`C compiler command`**

  CFLAGS — **`C compiler flags`**

  CXX — **`C++ compiler command`**

  CXXFLAGS — **`C++ compiler flags`**

  CPPFLAGS — **`C/C++ preprocessor flags,e.g. -I<incdir>`**
  if you have headers in a nonstandard directory **`<incdir>`**

  CPP — **`C preprocessor`**
- `` `make` `` is used to build the executables
  - Will reside in respective directories (e.g. **`nodet`** is in **`FUN3D_90`**)

Basic Operation

- Construct the *vanilla* **serial** executable
- Unpack your FUN3D distribution
  - Creates a directory "**`fun3d-12.7-74063`**"
- Enter the FUN3D distribution directory
- Run the `` `configure` `` script and build executables with `` `make` ``

```
$ mkdir serial
$ cd serial
$ ../configure
$ make
```

- Note that this will search for a supported compiler in your path
- Chooses the first one found based on pre-defined order
- Override this with the **`FC=mycompiler`** option
- MPI configurations will use the `` `--with-mpif90` `` wrapper if given

Did It Work? Expected Output

```
…
Configuration (FUN3D):

Source code location:  ..
Version:               12.7-74063
Fortran Compiler:      ifort
Fortran basis:         ifort
Fortran flags:         -O2 -ip -align
    -fno-alias -g -traceback
C Compiler:            gcc
C flags:               -g -O2
C++ Compiler:          g++
C++ flags:             -g -O2
Linker flags:          -lm
Dependencies:
build:
High Energy Physics:   no
Cmplx Variable Tools:  no
Python bindings:       no
FCCHT support:         no
FSI support:           no
PDF documentation:     yes

bindings:
Libcore:               internal
refine:                subpackage
CAPRI support:         no
```
page 1

```
knife:                 subpackage
MPI support:           no
CUDA support:          no
Zoltan:                no
ParMETIS:              no
Tecplot I/O:           no
6DOF libraries:        no
DiRTlib support:       no
SUGGAR support:        no
DYMORE support:        no
RCAS_SDX support:      no
CGNS support:          no
PORT support:          no
NPSOL support:         no
DOT support:           no
KSOPT support:         no
SNOPT support:         no
SMEMRD support:        version 1.3.1
IRS support:           no
SSDC support:          no
SFE support:           no
SPARSKIT support:      no
SBOOM support:         no
VisIt support:         no
```
page 2

- Executables created relative to the *serial* sub-directory
  - **FUN3D_90/nodet, Adjoint/dual, Design/opt_driver**

Extended Operation

(How we do it)

- Create a **parallel** version of the code
- Build in a separate *configuration* subdirectory
  - Stores object code and executables only
  - Does not pollute the source tree with object code
  - Multiple configurations utilize the same source

```
$ mkdir mpi
$ cd mpi
$ ../configure --with-mpi=/path/to/mpi \
               --with-parmetis=/path/to/parmetis
$ make
```

Did It Work? Expected Output

```
…
Configuration (FUN3D):

Source code location:  ..
Version:               12.7-74063
Fortran Compiler:      /path/to/mpi/bin/mpif90
Fortran basis:         ifort
Fortran flags:         -O2 -ip -align
    -fno-alias -g -traceback
C Compiler:            /path/to/mpi/bin/mpicc
C flags:               -g -O2
C++ Compiler:          g++
C++ flags:             -g -O2
Linker flags:          -lm
Dependencies:
build:
High Energy Physics:   no
Cmplx Variable Tools:  no
Python bindings:       no
FCCHT support:         no
FSI support:           no
PDF documentation:     yes

bindings:
Libcore:               internal
refine:                subpackage
CAPRI support:         no
```
page 1

```
knife:                 subpackage
MPI support:           no
CUDA support:          no
Zoltan:                no
ParMETIS:              /path/to/parmetis
Tecplot I/O:           no
6DOF libraries:        no
DiRTlib support:       no
SUGGAR support:        no
DYMORE support:        no
RCAS_SDX support:      no
CGNS support:          no
PORT support:          no
NPSOL support:         no
DOT support:           no
KSOPT support:         no
SNOPT support:         no
SMEMRD support:        version 1.3.1
IRS support:           no
SSDC support:          no
SFE support:           no
SPARSKIT support:      no
SBOOM support:         no
VisIt support:         no
```
page 2

- Executables created relative to the *mpi* sub-directory
  - **FUN3D_90/nodet, Adjoint/dual, Design/opt_driver**

Troubleshooting/FAQ (1 of 3)

fun3d-support@lists.nasa.gov

- Problems
  - "**checking for Fortran compiler default output file name... configure: error: Fortran compiler cannot create executables**
    See `config.log` for more details."
  - Make sure that Fortran compiler works in your environment
    - Adjust PATH, load appropriate GNU modules, MPI installation, etc.
  - Limited check of `configure` options
    - Bad "**--enable-***" and "**--with-***" options silently ignored
  - Option values containing spaces must be quoted from shell
    - e.g. **FCFLAGS="-g -O2 -m32 -fno-common"**
  - Do **NOT** configure in top level distribution directory and then try to make individual configuration directories
    - `**make distclean**` to clean a previous configuration of the source
  - Look/send "**config.log**" file
    - Also includes configuration options at the top (less quoted values w/ spaces)

Troubleshooting/FAQ (2 of 3)

fun3d-support@lists.nasa.gov

- Can I…
  - Override the default compiler options?
    - Yes, **--disable-ftune FCFLAGS="-what-ever-you-want"**
      - Remember some compilers always need certain options
  - Explicitly specify my compiler?
    - You can, with **FC=compiler**, but this will be overridden if using "**--with-mpif90**"
  - Fix anything by manually editing the `configure` script or Makefiles?
    - **NO**! and we cannot support any such action
    - Anything that you can safely change is governed by a configure option
  - Install the executables in a central location?
    - Yes, `make install` will install executables, etc. under the location given by the "**--prefix=/your/path**" option to `configure`

Troubleshooting/FAQ (3 of 3)

fun3d-support@lists.nasa.gov

- What if I…
  - Have a proprietary MPI installation?
    - Some HPC resources have proprietary MPI installations using non-standard paths and names
    - Use "**--with-mpibin**", "**--with-mpiinc**", "**--with-mpif90**", and "**--with-mpiexec**" along with their "**--without-***" counterparts as needed to specify the binary and include paths as well as the name for the `mpif90` compiler wrapper and, if needed, the `mpiexec` script
    - Paths can be absolute or relative to the "**--with-mpi**" and "**--with-mpibin**" values

```
$ ./configure --with-mpi=/path/to/mpi
              --with-mpif90=my_mpif90
              --without-mpiexec
              …
```

  - My MPI executables will not run
    - Check the consistency of your MPI compilation/runtime installations
    - The MPI installation used for compilation is available as MPI Prefix: from

```
$ /path/to/nodet/nodet_mpi --version
```

What We Covered

- How to configure and compile the FUN3D suite
  - Execute `` `configure` `` to localize a configuration
  - Build the executables with `` `make` ``
- Configuration options
  - Enable/Disable Features
  - With/Without Packages (3rd party libraries and tools)
  - Custom environment variables
- Use separate configuration subdirectories
  - Keeps source and object code separate
  - Allows multiple configurations under one source
  - Invoke as `` `../configure …` ``

FUN3D v13.1 Training

Session 4: Gridding, Solution, and Visualization Basics

Eric Nielsen

Learning Goals

What we will cover
- Basic gridding requirements and formats
- Nondimensionalizations and axis conventions
- Basic environment for running FUN3D
- FUN3D user inputs
- Running FUN3D for typical steady-state RANS cases
  - Compressible transonic turbulent flow over a wing-body using a tetrahedral VGRID mesh
  - Turbulent flow over a NACA 0012 airfoil section
- Things to help diagnose problems
- Visualization overview

What we will *not* cover
- Other speed regimes
- Unsteady flows

Gridding Considerations

- FUN3D is a **node-based** discretization
  - To get similar resolution when comparing with a cell-centered code, you must use a finer grid
    - E.g., on a tetrahedral grid, the grid for FUN3D must be ~2 times finer on the surface, and ~6 times finer in the volume mesh to be fair
  - This is critical when comparing with cell-centered solvers
  - Hanging nodes are not currently supported
- FUN3D integrates all of the way to the wall for turbulent flows
  - Wall function grids are not adequate
  - Goal is to place first grid point at y^+=1
    - Base Δy on a flat plate estimate using your Reynolds number; can examine result in solver output and tweak as necessary
- Users employ all of the common grid generators – VGRID, AFLR2/AFLR3/SolidMesh, ICEM, Pointwise, etc.
- FUN3D also supports point-matched, multiblock structured grids through Plot3D file input
  - Subject to certain grid topologies:
    - Singularities treated – i.e., hexes with collapsed faces converted to prisms
    - But hexes with 180° internal angles cause FUN3D discretization to break down (LSQ)
- FUN3D can convert tetrahedral VGRID meshes to mixed elements
- FUN3D can convert any mixed element grid into tetrahedra using command line option `'--make_tets'`

Supported Grid Formats

Grid Format	Formatted	Unformatted	Supports mixed elements	Direct load or converter	File extension(s)
FAST	X	X		Direct	.fgrid, .mapbc
VGRID (single or multisegment)		X		Direct	.cogsg, .bc, .mapbc
AFLR3	X	X Also Binary	X	Direct	.ugrid/.(l)r8.ugrid/.(l)b8.ugrid, .mapbc
FUN2D	X			Direct	.faces
Fieldview v2.4, v2.5, v3.0	X	X	X	Direct (Some details of format not supported)	.fvgrid_fmt, .fvgrid_unf, .mapbc
Felisa	X			Direct	.gri, .fro, .bco
Point-matched, multiblock Plot3D	X	X	Hexes, degenerates	Converter	.p3d, .nmf
CGNS		Binary	X	Converter	.cgns

The development team can work with you to handle other formats as needed

Boundary Condition Input File

- Where required, the FUN3D .mapbc file takes the form:

```
Number of boundary patches
Boundary patch index    BC index     Family name
```

- The BC index may be either a 4-digit FUN3D-style index or a GridTool-style index
- The family name is optional, but must be present if the user requests patch lumping by family

```
3
1  4000    Wing
2  5000    Farfield
3  6662    Symmetry plane
```

- Exception: The .mapbc format for VGRID meshes follows the GridTool/VGRID format

Nondimensionalization

- Notation: * indicates a dimensional variable, otherwise dimensionless; the reference flow state is ***usually*** free stream ("∞"), but need not be
- Define reference values:
  - L^*_{ref} = reference length of the physical problem (e.g. chord in ft)
  - L_{ref} = corresponding length in your grid (*dimensionless*)
  - ρ^*_{ref} = reference density (e.g. slug/ft^3)
  - μ^*_{ref} = reference molecular viscosity (e.g. slug/ft-s)
  - T^*_{ref} = reference temperature (e.g. ^{o}R, compressible only)
  - a^*_{ref} = reference sound speed (e.g. ft/s, compressible only)
  - U^*_{ref} = reference velocity (e.g. ft/s)
- Space and time are made dimensionless in FUN3D by:
  - $\vec{x} = \vec{x}^* /(L^*_{ref}/L_{ref})$ $\quad t = t^* a^*_{ref}/(L^*_{ref}/L_{ref})$ (compressible) $\quad t = t^* U^*_{ref}/(L^*_{ref}/L_{ref})$ (incompressible)

Nondimensionalization (cont)

- For the ***compressible flow*** equations the dimensionless variables are:
  - $\vec{u} = \vec{u}^* / a^*_{ref}$ so $|\vec{u}|_{ref} = |\vec{u}|^*_{ref} / a^*_{ref} = M_{ref}$
  - $P = P^* / (\rho^*_{ref} a^{*2}_{ref})$ so $P_{ref} = P^*_{ref} / (\rho^*_{ref} a^{*2}_{ref}) = 1/\gamma$
  - $a = a^* / a^*_{ref}$ so $a_{ref} = 1$
  - $T = T^* / T^*_{ref}$ so $T_{ref} = 1$
  - $e = e^* / (\rho^*_{ref} a^{*2}_{ref})$ so $e_{ref} = e^*_{ref} / (\rho^*_{ref} a^{*2}_{ref}) = 1/(\gamma(\gamma-1)) + M^2_{ref}/2$
  - $\rho = \rho^* / \rho^*_{ref}$ so $\rho_{ref} = 1$
  - From the equation of state and the definition of sound speed:

    $T = \gamma P / \rho = a^2$
- The input Reynolds number in FUN3D is related to the Reynolds number of the physical problem by

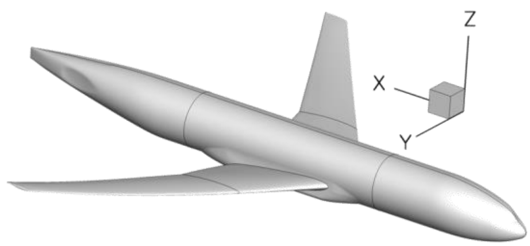
reynolds_number = $\mathrm{Re}_{ref} / L_{ref}$ where $\mathrm{Re}_{ref} = \rho^*_{ref} U^*_{ref} L^*_{ref} / \mu^*_{ref}$

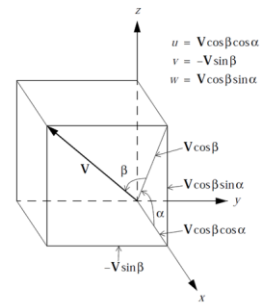
i.e. reynolds_number is a Reynolds number ***per unit grid length***

Setting the Reynolds Number Input

- Frequent cause of confusion, even for developers
- Need to know what characteristic length your Reynolds number is based on – mean aerodynamic chord, diameter, etc.
- Your input Reynolds number is based on the corresponding length of that “feature” in your computational grid
- Example: You want to simulate a Reynolds number of 2.5 million based on the MAC:
  - If the length of the MAC in your grid is 1.0 grid units, you would input Re=2500000 into FUN3D
  - If the length of the MAC in your grid is 141.2 grid units (perhaps these physically correspond to millimeters), you would input 2500000/141.2, or Re=17705.4 into FUN3D

FUN3D Axis Convention





- FUN3D coordinate system differs from the standard wind coordinate system by a 180º rotation about the y-axis
  - Positive x-axis is toward the “back” of the vehicle (downstream)
  - Positive y-axis is out the “right wing”
  - Positive z-axis is “upward”
- The freestream angle of attack and yaw angle are defined as shown

Runtime Environment

- “Unlimit” your shell (also good idea to put this in any queue scripts):

```
$ ulimit unlimited # for bash
$ unlimit          # for c shell
```

- If unformatted or binary, what “endianness” does your grid file have?
  - E.g., VGRID files are always big endian, regardless of platform
  - If your compiler supports it, FUN3D will attempt to open files using an `open(convert=…)` syntax
  - Most compilers support some means of conversion
    - Either an environment variable or compile-time option, depending on what compiler you’re using
    - E.g., Intel compiler can be controlled with an environment variable `F_UFMTENDIAN = big`
- Memory required by solver: *rough* rule of thumb is 3-3.5 GB per million points (not cells!)
  - Conversely, 200k-300k points per 1 GB of memory
    - Users generally partition into smaller domains than this, but be aware of these numbers
  - This memory estimate will be higher if visualization options are used, etc

User Inputs for FUN3D

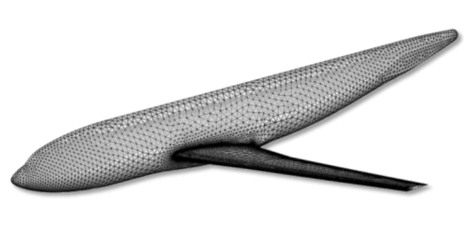
Input deck `fun3d.nml`

- The user is required to supply an input deck for FUN3D named `fun3d.nml` (fixed name)
- This filename contains a collection of Fortran namelists that control FUN3D execution – all namelist variables have default values as documented
- But user will need to set at least some high-level variables, such as the project name

Command Line Options (CLOs)

- CLOs always take the form `--command_line_option` after the executable name
  - Some CLOs may require trailing auxiliary data such as integers and/or reals
- User may specify as many CLOs as desired
- CLOs always trump `fun3d.nml` inputs
- CLOs available for a given code in the FUN3D suite may be viewed by using `--help` after the executable name
- Most CLOs are for developer use; namelist options are preferred where available

Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh



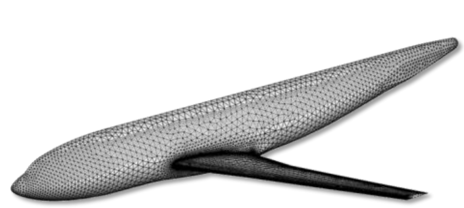
- For this case, we will assume that someone has provided a set of VGRID files containing the mesh
  - f6fx2b_trn.cogsg, f6fx2b_trn.bc, and f6fx2b_trn.mapbc
- It is always a good idea to examine the .mapbc file first to check the boundary conditions and any family names
  - Note that specific boundary conditions will be covered in a separate session

Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh

- For this case, the VGRID/GridTool-style .mapbc file is as shown
- Surface grid consists of 51 patches
- Note that VGRID/GridTool-style BC's are specified
- Family names are also as shown (required in this format)
- FUN3D does not use the other columns of data
- If you cannot easily visualize your mesh to set appropriate boundary conditions, one easy approach is to set them all to inflow/outflow, then run a single time step of FUN3D with boundary visualization activated – then set patch BC's as needed for actual simulation

```
#Thu Mar 11 13:42:40 2010
#bc.map
Patch #        BC             Family   #surf   surfIDs        Family
#------------------------------------------------------------------------
1              3              3             0          0           Box
2              3              3             0          0           Box
3              3              3             0          0           Box
4              3              3             0          0           Box
5              3              3             0          0           Box
6              4              4             1          15          Wing
7              4              4             1          15          Wing
8              4              4             1          17          Wing
9              4              4             1          17          Wing
10             4              4             1          15          Wing
11             4              4             1          13          Fuselage
12             4              4             1          21          Fuselage
13             4              4             1          11          Fuselage
14             4              4             1          11          Fuselage
15             4              4             1          12          Fuselage
16             4              4             1          12          Fuselage
17             4              4             1          15          Wing
18             4              4             1          15          Wing
19             4              4             1          15          Wing
20             4              4             1          15          Wing
21             4              4             1          17          Wing
22             4              4             1          17          Wing
23             4              4             1          16          Wing
24             4              4             1          15          Wing
25             4              4             1          17          Wing
26             4              4             1          8           Fuselage
27             4              4             1          16          Wing
28             4              4             1          16          Wing
29             4              4             1          16          Wing
30             4              4             1          16          Wing
31             4              4             1          18          Wing
32             4              4             1          18          Wing
33             4              4             1          17          Wing
34             4              4             1          18          Wing
35             4              4             1          18          Wing
36             4              4             1          1           Wing
37             4              4             1          18          Wing
38             4              4             1          18          Wing
39             4              4             1          18          Wing
40             4              4             1          22          Fuselage
41             1              1             0          0           Symmetry
42             4              4             1          10          Fuselage
43             4              4             1          9           Fuselage
44             4              4             1          14          Fuselage
45             4              4             1          23          Fuselage
46             4              4             1          19          Wing
47             4              4             1          20          Wing
48             4              4             1          27          Fairing
49             4              4             1          29          Fairing
50             4              4             1          28          Fairing
51             4              4             1          30          Fairing
```


Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh



- Now we will look at the minimum set of user inputs needed in `fun3d.nml` to run this case

Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh

```
&project
  project_rootname = 'f6fx2b_trn'          Project name
/
&raw_grid
  grid_format = 'vgrid'                    Read a set of VGRID files
/
&reference_physical_properties
  mach_number       = 0.75                 Sets freestream Mach number
  reynolds_number   = 17705.40             Sets Reynolds number
  angle_of_attack   = 1.0                  Sets freestream angle of attack
  temperature       = 580.0                Sets freestream temperature
  temperature_units = "Rankine"            Uses Rankine temperature units for input
/
&code_run_control
  restart_read = 'off'                     Perform a cold start
  steps        = 500                       Perform 500 time steps
/
&force_moment_integ_properties
  area_reference   = 72700.0               Sets reference area
  x_moment_length = 141.2                  Sets length for normalizing y-moments
  y_moment_length = 585.6                  Sets length for normalizing x-, z-moments     All in grid units
  x_moment_center = 157.9                  Sets x-moment center
  z_moment_center = -33.92                 Sets z-moment center
/
&nonlinear_solver_parameters
  schedule_cfl     = 10.0 200.0            CFL for meanflow is ramped from 10.0 to 200.0
  schedule_cflturb = 1.0 30.0              CFL for turbulence is ramped from 1.0 to 30.0
/
```


Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh

- We now have the boundary conditions and input deck set up to run FUN3D
- To execute FUN3D, we use the following basic command line syntax:

  `mpirun ./nodet_mpi`

  – Note your environment may require slightly different syntax:
    - `mpirun` vs `mpiexec` vs `aprun` vs …
    - May need to specify various MPI runtime options:
      - `-np #`
      - `-machinefile filename`
      - `-nolocal`
      - Others

Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh

- Using 1 Intel Haswell node (24 cores), this case runs in 2-3 minutes
- The top of the screen output will include an echo of your `fun3d.nml`, as well as some preprocessing information:

```
FUN3D 12.7-74063 Flow started 05/18/2015 at 06:09:15 with 24 processes
       [Echo of fun3d.nml]
 The default "unformatted" data format is being
   used for the grid format "vgrid".
... nsegments,ntet,nnodesg            1     2994053      513095
cell statistics: type,      min volume,      max volume, max face angle
cell statistics:  tet,  0.41152313E-06,  0.66593449E+11,  179.973678915
cell statistics:  all,  0.41152313E-06,  0.66593449E+11,  179.973678915

     ... PM (64,skip_do_min) :            0 F
     ... Calling ParMetis (ParMETIS_V3_PartKway) ....           0 F
     ... edgeCut       140453
     ... Time for ParMetis: .2 s
     ... Constructing partition node sets for level-0...               2994053 T
     ... Edge Partitioning ....
     ... Boundary partitioning....
     ... Reordering for cache efficiency....
     ... Write global grid information to f6fx2b.grid_info
     ... Time after preprocess TIME/Mem(MB):      1.60    180.52    180.52
  NOTE: kappa_umuscl set by grid: .00

 Grid read complete
  Repaired 82 nodes of symmetry plane 6662, max deviation: 0.172E-03
  y-symmetry metrics modified/examined: 23601/23601
  Distance_function unique ordering  T    20000000
  construct partial boundary...nloop=           1
  find closer surface edge...
  find closer surface face...

 Wall spacing: 0.766E-03 min, 0.120E-02 max, 0.115E-02 avg
```

FUN3D version, start time, job size

VGRID input is being used
Grid contains 2,994,053 tets and 513,095 points
Min/max cell volumes, max internal face angles

# of edges cut by partitioning (measure of communication)

1.6 secs required to preprocess the mesh

Min/max/avg wall spacing statistics

Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh

- At this point, time stepping commences
- For each time step:
  - The L2-norm of the density|turbulence equation is red|blue; max and location are also included
  - Lift and drag are reported in green
- "`Done.`" indicates execution is complete

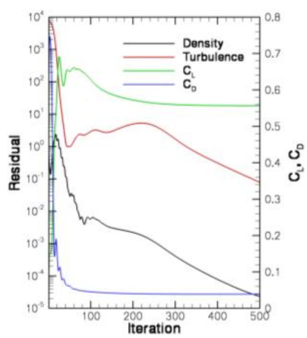
```
Iter             density_RMS  density_MAX   X-location   Y-location   Z-location
                    turb_RMS     turb_MAX   X-location   Y-location   Z-location
   1  0.567454200028342E+00  0.28035E+02  0.16377E+03 -0.16562E+03  0.20117E+02
      0.764159584901741E+04  0.13249E+07  0.79654E+04 -0.88280E+04  0.25675E+02
      Lift  0.103222129717669E+00         Drag  0.646514468368827E+00
   2  0.300676687726037E+00  0.12718E+02  0.29226E+03 -0.72487E+02 -0.12411E+02
      0.753354469872627E+04  0.12868E+07  0.79654E+04 -0.88280E+04  0.25675E+02
      Lift  0.146830367737086E+00         Drag  0.721243419758588E+00
 .
 .
 .
 499  0.235098406158263E-04  0.44827E-02  0.63496E+04 -0.38199E+04  0.18712E+04
      0.799698877237297E-01  0.12961E+02  0.46732E+04 -0.15204E+04  0.26710E+03
      Lift  0.556610229549889E+00         Drag  0.388376897833650E-01
 500  0.232908407834686E-04  0.44201E-02  0.63496E+04 -0.38199E+04  0.18712E+04
      0.789246351974423E-01  0.12785E+02  0.46732E+04 -0.15204E+04  0.26710E+03
      Lift  0.556607946389416E+00         Drag  0.388374809483346E-01

Writing f6fx2b_trn.flow (version 11.8) lmpi_io 2
 inserting current history iterations 500
Time for write: .0 s

Done.
```

Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh

- FUN3D provides a couple of text files with basic statistics and summary data:
  - `f6fx2b_trn.grid_info` File containing basic mesh statistics and partitioning info
  - `f6fx2b_trn.forces` File containing force breakdowns by boundary and totals
- FUN3D also produces:

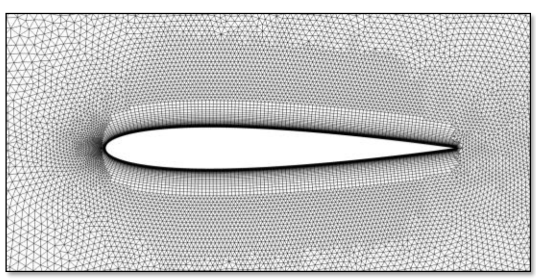
  `f6fx2b_trn_hist.dat` Tecplot file with residual, force convergence histories

  `f6fx2b_trn.flow` Solver restart information



- For this particular case, the mean flow and turbulence residuals are reduced by ~5 orders of magnitude over 500 time steps
- Lift and drag come in after a few hundred time steps

NACA 0012 Airfoil



- For this case, we have been given a set of binary, big endian AFLR3 files
  - 0012.b8.ugrid, 0012.mapbc
  - For computations in 2D mode
    - Grid must be one-element wide in the y-direction (except when using FUN2D format)
    - Grid must contain only prisms and/or hexes
- First check the .mapbc file
  - The y-planes must be separate boundary patches and should be given BC 6662

0012.mapbc

```
4
1 4000
2 5000
3 6662
4 6662
```

NACA 0012 Airfoil

- `fun3d.nml` is shown here
- FUN2D grid format will automatically be executed in 2D mode; all others must be explicitly put in 2D mode

```
&project
  project_rootname = '0012'
/
&raw_grid
  grid_format = 'aflr3'
  twod_mode   = .true.
/
&reference_physical_properties
  mach_number       = 0.80
  reynolds_number   = 1.e6
  angle_of_attack   = 1.25
  temperature       = 580.0
  temperature_units = "Rankine"
/
&code_run_control
  restart_read = 'off'
  steps        = 5000
/
&force_moment_integ_properties
  area_reference  = 0.1
  x_moment_center = 0.25
/
&nonlinear_solver_parameters
  schedule_cfl     = 10.0 200.0
  schedule_cflturb = 1.0 10.0
/
&global
  boundary_animation_freq = -1
/
```

Read an AFLR3 grid
Execute in 2D mode

NACA 0012 Airfoil

```
FUN3D 12.7-74063 Flow started 05/18/2015 at 09:06:46 with 24 processes
        [Echo of fun3d.nml]
 The default "stream" data format is being
   used for the grid format "aflr3".
Preparing to read binary AFLR3 grid: 0012.b8.ugrid

  nnodes              116862
  ntface,nqface       204510 14607
  ntet,npyr,nprz,nhex 0 0 102255 7047

cell statistics: type,      min volume,      max volume, max face angle
cell statistics:  prz,  0.16960303E-06,  0.52577508E-01,  164.861624007
cell statistics:  hex,  0.83173480E-09,  0.12843645E-04,  123.906431556
cell statistics:  all,  0.83173480E-09,  0.52577508E-01,  164.861624007

 ... PM (64,skip_do_min) :            0 F
     ... Calling ParMetis (ParMETIS_V3_PartKway) ....          0 F
     ... edgeCut        11490
     ... Time for ParMetis: .1 s
     ... checking for spanwise edge cuts.
     ... Constructing partition node sets for level-0...               109302 T
     ... Edge Partitioning ....
     ... Boundary partitioning....
     ... Euler numbers Grid:1 Boundary:0 Interior:0
     ... Reordering for cache efficiency....
     ... ordering edges for 2D.
     ... Write global grid information to 0012.grid_info
     ... Time after preprocess TIME/Mem(MB):      0.31      90.82      90.82
  NOTE: kappa_umuscl set by grid: .00

 Grid read complete

 Using 2D Mode (Node-Centered)

Distance_function unique ordering  T    20000000
  construct partial boundary...nloop=          1
  find closer surface edge...
  find closer surface face...
 Wall spacing: 0.100E-03 min, 0.100E-03 max, 0.100E-03 avg
```

Binary AFLR3 format is the default

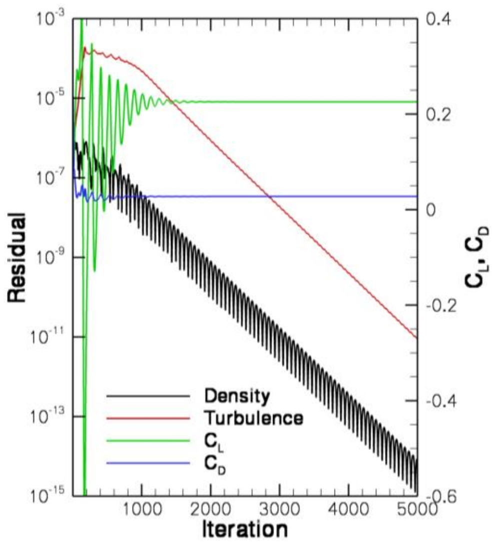
Binary AFLR3 grid being read

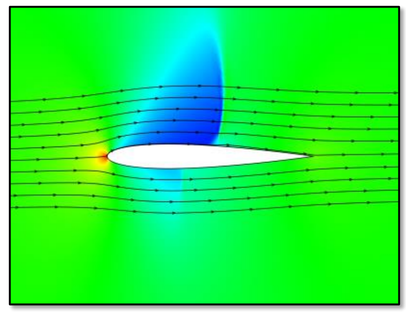
Grid contains 116,862 points
Grid contains 204,510 tris, 14,607 quads
Grid contains 102,255 prisms, 7,047 hexes

Cell stats now broken out by cell type

Solver running in 2D mode

NACA 0012 Airfoil





List of Key Input/Output Files

- Input
  - Grid files (prefixed with project name, suffixes depend on grid format)
  - `fun3d.nml`
- Output
  - `[project].grid_info`
  - `[project].forces`
  - `[project]_hist.dat`
  - `[project].flow`

What Could Possibly Go Wrong?

Problem

- Common complaint from VGRID meshes during initial preprocessing phase at front end of solver:

```
Checking volume-boundary connectivity...

stopping...unable to find common element for face       1 of
boundary       3
boundary nd array      46  17368 334315

node,locvc          46******************************
node,locvc_type     46    tet    tet    tet    tet    tet

node,locvc       17368************************************
node,locvc_type  17368    tet    tet    tet    tet    tet    tet
```

- This is due to a very old VGRID bug that causes an incompatibility between the `.cogsg` and `.bc` files
  - Compile and run `utils/repair_vgrid_mesh.f90` to generate a valid `.bc` file to replace your original one

What Could Possibly Go Wrong?

Problem

- Common complaint from unformatted/binary meshes during initial preprocessing phase at front end of solver:

```
Read/Distribute Grid.
forrtl: severe (67): input statement requires too much data, unit 16100,
file /misc/work14/user/FUN3D/project.cogsg
```

- Check the endianness of the grid and your environment/executables

Problem

- Unexpected termination, especially during preprocessing or first time step
  - Are your shell limits set?
  - Do you have enough local memory for what you are trying to run?

What Could Possibly Go Wrong?

Problem

- Solver diverges or does not converge
  - Problem-dependent, very tough to give general advice here
  - Sometimes require first-order iterations (primarily for high speeds)
  - Sometimes require smaller CFL numbers
  - Sometimes require alternate flowfield initialization (not freestream) in some subregion of the domain: e.g., chamber of an internal jet
  - Check your boundary conditions and gridding strategy
  - Perhaps your problem is simply unsteady

Problem

- Solver suddenly dies during otherwise seemingly healthy run
  - Sometimes useful to visualize solution just before failure
  - Is it a viscous case on a VGRID mesh? Try turning on `large_angle_fix` in `&special_parameters` namelist (viscous flux discretization degenerates in sliver cells common to VGRID meshes)
  - Is it a turbulent flow on a mesh generated using AFLR3? Look for “eroded” boundary layer grids near geometric singularities – AFLR3 sometimes has trouble adding viscous layers near complex corners, etc

What Could Possibly Go Wrong?

In General…

- Do not hesitate to send questions to *fun3d-support@lists.nasa.gov* ; we are happy to try to diagnose problems
  - Please send as much information about the problem/inputs/environment that you can, as well as all screen output, any error output, and `config.log`
  - In extreme cases, we may request your grid and attempt to run a case for you to track down the problem
  - If you cannot send us a case due to restrictions, size, etc, a generic/smaller representative case that behaves similarly can be useful
  - Check the manual for guidance
- Ask the FUN3D user community, *fun3d-users@lists.nasa.gov*

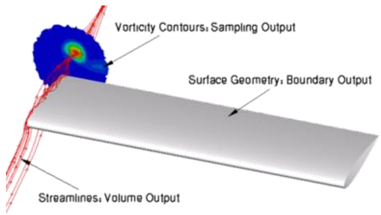
Visualization Learning Goals

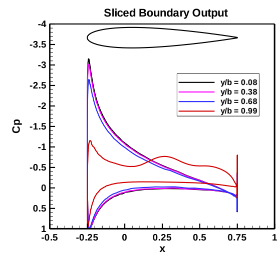
- What this will teach you
  - Run-time flow visualization output
    - Output on boundary surfaces
    - Output on user-specified “sampling” surfaces within the volume
    - Output of full volume data
    - Output generated by “slicing” boundary data - “sectional” output
- What you will not learn
  - The plethora of output options available for visualization
  - Tecplot usage
- What should you already know
  - Basic flow solver operation and control

Background

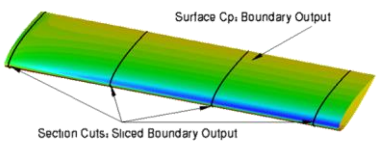
- Datasets are getting simply too large to post-process in a traditional manner
- FUN3D allows visualization data to be generated as the solver is running
    - User specified frequency and output type
    - User specified output variables from a fairly extensive list
- Majority of output options are Tecplot-based
  - Volume output may also be generated in Fieldview, CGNS formats
- Note FUN3D also supports true in-situ visualization at scale using the DoE VisIt package; however, this is not covered here
  - Intelligent Light is currently integrating VisIt’s in-situ capabilities with Fieldview

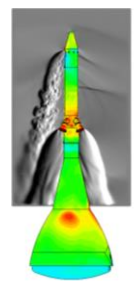
Selected Visualization Output Examples





Iso-surfaces





Schlieren,
boundary output

Visualization Overview

- All of the visualization outputs require similar namelist-specified “frequency” N to activate:
  - In all cases, N = 0, 1, 2, 3, …
    - N = 0 generates no output
    - N < 0 generates output only at the ***end*** of the run - typically used for steady-state cases. The actual value of N is ignored
    - N > 0 generates output every Nth time step - typically used to generate animation for unsteady flows; can also be used to observe how a steady flow converges

Visualization Overview

- Customizable output variables (except sliced boundary data):
  - Most variables are the same between the boundary surface, sampling and volume output options; boundary surface has a few extra
  - See manual for lists of all available variables
  - Default variables always include x, y, z, and the "primitive" flow variables u, v, w, and p (plus density if compressible)
  - Several "shortcut" variables: e.g.,
    `primitive_variables = rho, u, v, w, p`
  - Must explicitly turn off the default variables if you don't want them (e.g., **`primitive_variables = .false.`**)
  - Variable selection for each coprocessing option done with a different namelist to allow "mix and match"

Visualization Overview

- For boundary surface output, default is all solid boundaries in 3D and one y=const plane in 2D; alternate output boundaries selected with, e.g.:

```
&boundary_output_variables
   number_of_boundaries = 3
   boundary_list = '3,5,9'    ! blanks OK as
                                delimiter too: '3 5 9'
                              ! dashes OK as delimiter
                                too: '3-9'
 /
```

- If you already have a converged solution and don't want to advance the solution any further, can do a "pass through" run:
  - set **`steps = 0`** in **`&code_run_control`**
  - You must have a restart file (**`[project].flow`**)
  - Run the solver with the appropriate namelist input to get desired output
  - **`[project].flow`** will remain unaltered after completion

Visualization Overview

- Sampling output requires additional data to describe the desired sampling surface(s)
  - Specified in namelist **`&sampling_parameters`**
  - Surfaces may be planes, quadrilaterals or circles of arbitrary orientation, or may be spheres or boxes
  - Isosurfaces and schlierens also available
  - Points may also be sampled
  - See manual for complete info
- Sliced boundary surface output requires additional data to describe the desired slice section(s)
  - Specified in namelist **`&slice_data`**
  - Always / only outputs x, y, z, C_p, C_{fx}, C_{fy}, C_{fz}
  - User specifies which (solid) boundaries to slice, and where
  - See manual for complete info

Visualization Overview

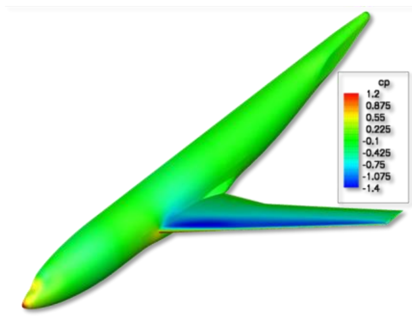
- Output files will be ASCII unless you have built FUN3D against the Tecplot library (exception: sliced boundary data is always ASCII)
  - ASCII files have .dat extension
  - Binary files have .plt extension - smaller files; load into Tecplot faster
  - Boundary output file naming convention (T = time step counter):
    - **`[project]_tec_boundary_timestepT.dat`** if N > 0
    - **`[project]_tec_boundary.dat`** if N < 0
  - Volume output file naming convention (note: 1 file *per processor* P)
    - **`[project]_partP_tec_volume_timestepT.dat`** if N > 0
    - **`[project]_partP_tec_volume.dat`** if N < 0
  - Sampling output file naming convention (one file per sampling geometry G):
    - **`[project]_tec_sampling_geomG_timestepT.dat`** if N > 0
    - **`[project]_tec_sampling_geomG.dat`** if N < 0

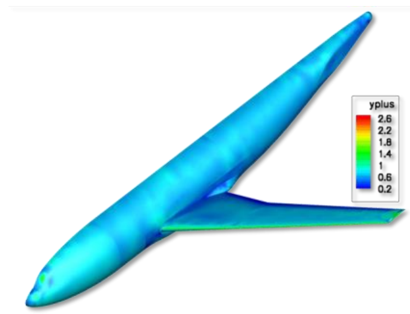
Boundary Output Visualization Example

```
&global
  boundary_animation_freq = -1
/
&boundary_output_variables
  primitive_variables = .false.
  cp                  = .true.
  yplus               = .true.
/
```

Dump boundary vis at end of run

Turn off rho, u, v, w, p
Turn on C_p
Turn on y^+



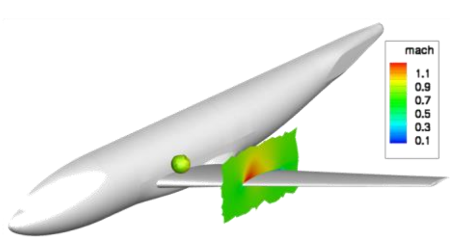


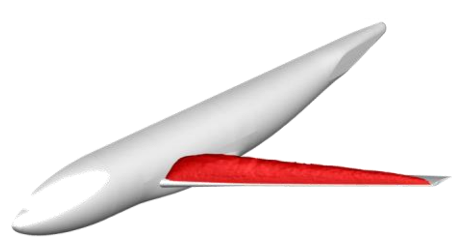
Sampling Visualization Example

```
&sampling_parameters
  number_of_geometries  = 3
  type_of_geometry(1)   = 'plane'
  plane_center(2,1)     = -234.243
  plane_normal(2,1)     = 1.0
  sampling_frequency(1) = -1
  type_of_geometry(2)   = 'sphere'
  sphere_center(1,2)    = 74.9
  sphere_center(2,2)    = -107.7
  sphere_center(3,2)    = 50.0
  sphere_radius(2)      = 20.0
  sampling_frequency(2) = -1
  type_of_geometry(3)   = 'isosurface'
  isosurf_variable(3)   = 'mach'
  isosurf_value(3)      = 1.00
  sampling_frequency(3) = -1
/
&sampling_output_variables
  primitive_variables = .false.
  mach                = .true.
/
```

Want 3 sampling geometries
First geometry is a plane
Plane y-coordinate
Plane y-normal
Write at end of run
Second geometry is a sphere
Center x-coordinate
Center y-coordinate
Center z-coordinate
Sphere radius
Write at end of run
Third geometry is an isosurface
Isosurface will be based on Mach number
Isosurface defined by Mach=1
Write at end of run

Turn off rho, u, v, w, p
Turn on Mach number



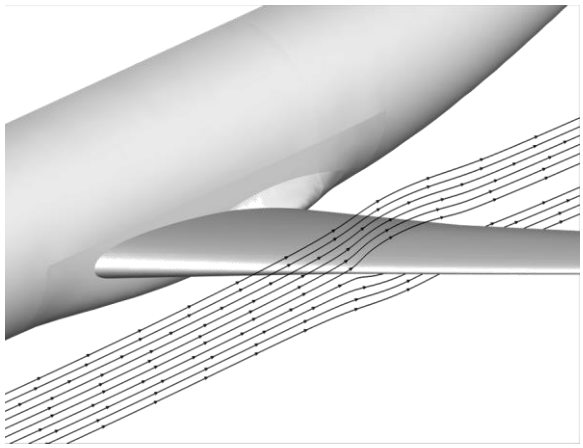


Volume Visualization Example

```
&global
  volume_animation_freq = -1
/
&volume_output_variables
  export_to='tecplot'
/
```

Dump output at end of run

Send results to Tecplot file



http://fun3d.larc.nasa.gov

FUN3D Training Workshop
July 30, 2017

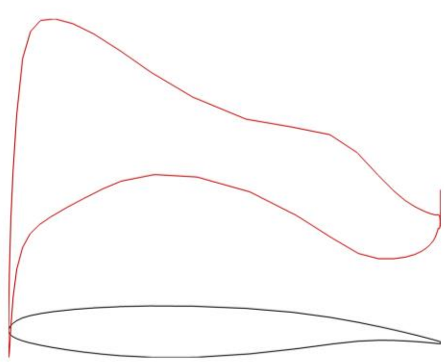
39

Slicing Visualization Example

```
&global
  slice_freq = -1
/
&slice_data
  nslices = 1
  slice_location(1) = -234.243
/
```

Dump output at end of run

Perform one slice
Coordinate of slice



http://fun3d.larc.nasa.gov

FUN3D Training Workshop
July 30, 2017

40

Troubleshooting/FAQ

- I can see what look like ragged dark lines on sampling surfaces and volume data – what is that?
  - Duplicate information at partition boundaries is not removed; if surface is not completely opaque, double plotting locally doubles the opaqueness (duplicate info ***is*** removed from boundary surface output)
  - Turn off transparency in Tecplot
- When I dump out volume plot files in Tecplot format, I get a file for every processor – is there a way around this?
  - Yes, in `&volume_output_variables` add: `export_to = 'tec'`
  - The team is working with Tecplot to develop their next generation of I/O APIs, with special focus on massively parallel needs
  - Alternative: switch to Fieldview or CGNS output, which uses a single file

What We Learned

- Basic gridding requirements and file formats
- Runtime environment
- How to set up boundary conditions and very basic FUN3D input decks
- How to run a tetrahedral RANS solution for a wing-body VGRID mesh
- How to perform a 2D mixed element airfoil solution using an AFLR3 grid
- Some unhealthy things to watch for and possible remedies
- Overview of visualization output options and examples

Don't hesitate to send questions our way!
fun3d-support@lists.nasa.gov

FUN3D v13.1 Training

Session 5:
Adjoint-Based Design for Steady Flows

Eric Nielsen

Learning Goals

- Introduction and basic approach taken in FUN3D
- Some lingo/nomenclature
- What is an adjoint, and what is it used for?
  - Error estimation and mesh adaptation
  - Sensitivity analysis for design optimization
- Design variables
- Objective/constraint functions
- Geometry parameterizations
- Setup and execution of a simple unconstrained problem
- Things to watch out for
- How to interpret results

What we will *not* cover
- Body transforms, body grouping
- Overset grid details
- Multipoint/multiobjective/constrained optimization
- Hooking in your own optimizer, parameterization tools
- Forward-mode differentiation using complex variables
- Design of unsteady flows
  - Later session

What to Expect

- Cost of design optimization is very problem-dependent, but in general you can expect to spend ~20 times the cost of a flow solution to get reasonable improvements, depending on how "good" the baseline is
- Generally see very rapid improvements initially, followed by diminishing returns
- We will cover the bare essentials here; also see the manual
  - There are many aspects we will not have time to cover here
- Hands-off design is challenging – be patient, send in questions, and we'll try to help you through
  - There are a lot of pieces involved, and getting things running smoothly always involves stumbling blocks along the way

Design Optimization Using FUN3D

- Based on a gradient-based approach
- FUN3D is distributed with support for several COTS gradient-based optimization packages
  - You must download and install your choice of these third-party libraries
    - DOT/BIGDOT (Vanderplaats R&D)
    - KSOPT (Greg Wrenn @ Langley)
    - PORT (Bell Labs)
    - NPSOL (Stanford)
    - SNOPT (Stanford)
    - Other packages are generally straightforward to hook up – couple of hours
- These optimizers are based on the user supplying functions and gradients (and perhaps constraints and their gradients also)
  - Optimizers know nothing about CFD, all they see are f and ∇f
- In CFD, objective/constraint functions are generally based on things like lift, drag, pitching moment, etc.
  - But can be anything you code up, generally speaking

Design Optimization Components

Functions

- When the optimizer requests a function value, it requires a flow solution with inputs and a grid corresponding to the current design variables

Gradients

- When the optimizer requests a gradient value, it requires a sensitivity analysis with inputs and a grid corresponding to the current design variables
  - The most straightforward way to generate sensitivity information is to perturb each design variable independently and run black-box finite differences
    - This is prohibitively expensive when each finite difference requires a new CFD simulation (or two) – cost scales linearly with the number of design variables
  - The most efficient sensitivity analysis approach for CFD simulations based on large numbers of design variables (hundreds or thousands) is the adjoint method

Notation and Governing Equations

We wish to perform rigorous adaptation and design optimization based on the steady-state Euler/Navier-Stokes equations, ***without requiring any a priori knowledge of the problem***:

$$\frac{\partial \mathbf{Q}}{\partial t} + \mathbf{R}(\mathbf{D}, \mathbf{Q}, \mathbf{X}) = 0$$

$\mathbf{R}$ = Spatial residual $\mathbf{Q}$ = Dependent variables

$\mathbf{D}$ = Design variables $\mathbf{X}$ = Computational grid

- Incompressible through hypersonic flows
- May include turbulence models and various physical models from perfect gas through thermochemical nonequilibrium

What is an Adjoint?

Combine cost function with Lagrange multipliers $\mathbf{\Lambda}$:

$$L(\mathbf{D},\mathbf{Q},\mathbf{X},\mathbf{\Lambda}_f,\mathbf{\Lambda}_g) = \underbrace{f(\mathbf{D},\mathbf{Q},\mathbf{X})}_{\text{Cost Function}} + \mathbf{\Lambda}_f^T \underbrace{\mathbf{R}(\mathbf{D},\mathbf{Q},\mathbf{X})}_{\text{Flowfield Equations}} + \mathbf{\Lambda}_g^T \underbrace{(\mathbf{K}\mathbf{X}-\mathbf{X}_{surf})}_{\text{Mesh Movement Equations}}$$

f = Cost function (lift/drag/boom/etc)

$\mathbf{K}$ = Mesh movement elasticity matrix

$\mathbf{\Lambda}_f$ = Flowfield adjoint variable

$\mathbf{\Lambda}_g$ = Grid adjoint variable

Differentiate with respect to $\mathbf{D}$:

$$\frac{dL}{d\mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{D}}\right]^T \mathbf{\Lambda}_f + \left[\frac{\partial \mathbf{Q}}{\partial \mathbf{D}}\right]^T \left\{\frac{\partial f}{\partial \mathbf{Q}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]^T \mathbf{\Lambda}_f\right\}$$
$$+ \left[\frac{\partial \mathbf{X}}{\partial \mathbf{D}}\right]^T \left\{\frac{\partial f}{\partial \mathbf{X}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right]^T \mathbf{\Lambda}_f + \mathbf{\Lambda}_g^T \mathbf{K}\right\} - \mathbf{\Lambda}_g^T \left[\frac{\partial \mathbf{X}}{\partial \mathbf{D}}\right]_{surf}$$

$$\rightarrow \underbrace{\left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]^T}_{\text{Governing Eqns}} \mathbf{\Lambda}_f = -\underbrace{\frac{\partial f}{\partial \mathbf{Q}}}_{\text{Engineering Output}}$$

This adjoint equation for the flowfield has powerful implications for:

- Error estimation & mesh adaptation
- Sensitivity analysis

Adjoints for Error Estimation and Mesh Adaptation

It is apparent that:

$$\mathbf{\Lambda}_f \equiv \frac{\partial f}{\partial \mathbf{R}} \longrightarrow$$

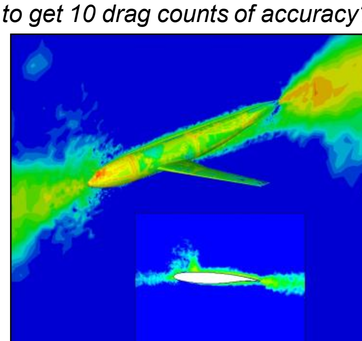
Direct relationship between local equation error and the output we are interested in

- These relationships can be used to get error estimates on f
- Also used to compute a scalar field explicitly relating local point spacing requirements to output accuracy for a user-specified error tolerance
- Often yields non-intuitive insight into gridding requirements
- Relies on underlying mathematics to adapt, rather than heuristics such as solution gradients

User no longer required to be a CFD expert to get the right answer

Transonic Wing-Body:

"Where do I need to put grid points to get 10 drag counts of accuracy?"



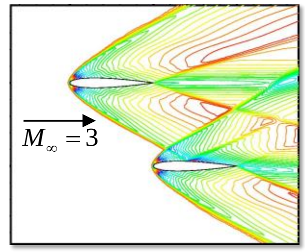
Blue=Sufficient Resolution

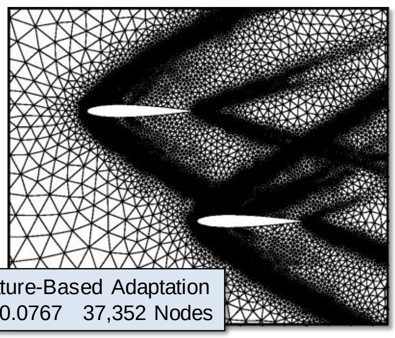
Red=Under-Resolved

Supersonic Adjoint-Based Mesh Adaptation

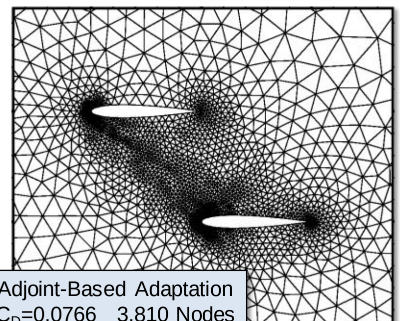
Collaboration with Venditti/Darmofal of MIT using FUN2D

- **Objective:** Adapt grid to compute drag on lower airfoil as accurately as possible
- **Result of adjoint-based adaptation:**
  - Uniformly-resolved shocks are not required
  - Drag is computed accurately with a <u>90% smaller grid</u>





Feature-Based Adaptation
C_D=0.0767 37,352 Nodes

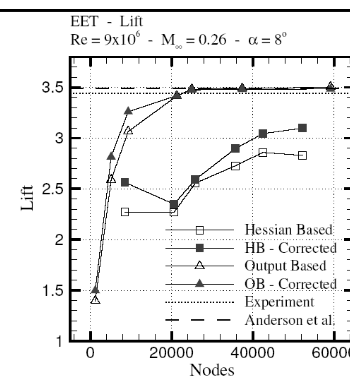


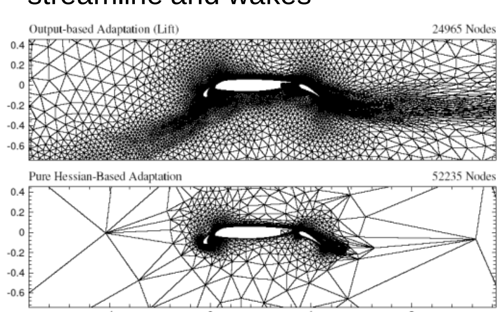
Adjoint-Based Adaptation
C_D=0.0766 3,810 Nodes

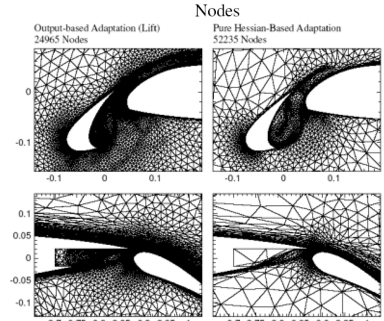
Adjoint-Based Mesh Adaptation for High Lift

Collaboration with Venditti/Darmofal of MIT using FUN2D

- Initial grid was coarse Euler mesh
- Pressure-based indicator only resolves strong flow curvature
- Adjoint-based indicator also includes important smooth regions, stagnation streamline and wakes







Adjoints for Sensitivity Analysis

Examine the remaining terms in the linearization:

$$\frac{dL}{d\mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{D}}\right]^T \mathbf{\Lambda}_f + \left[\frac{\partial \mathbf{X}}{\partial \mathbf{D}}\right]^T \left\{ \frac{\partial f}{\partial \mathbf{X}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right]^T \mathbf{\Lambda}_f + \mathbf{\Lambda}_g^T \mathbf{K} \right\} - \mathbf{\Lambda}_g^T \left[\frac{\partial \mathbf{X}}{\partial \mathbf{D}}\right]_{surf}$$

$$\rightarrow \quad \mathbf{K}^T \mathbf{\Lambda}_g = -\left\{ \frac{\partial f}{\partial \mathbf{X}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right]^T \mathbf{\Lambda}_f \right\}$$

Discrete adjoint equation for mesh movement

$$\rightarrow \quad \frac{dL}{d\mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \mathbf{\Lambda}_f^T \frac{\partial \mathbf{R}}{\partial \mathbf{D}} - \mathbf{\Lambda}_g^T \left[\frac{\partial \mathbf{X}}{\partial \mathbf{D}}\right]_{surf}$$

Sensitivity equation

Function Evaluation

1. Compute surface mesh at current D
2. Solve mesh movement equations
3. Solve flowfield equations

Sensitivity Evaluation

3. Solve flowfield adjoint equations
2. Solve mesh adjoint equations
1. Matrix-vector product over surface

Analysis Cost = Sensitivity Analysis Cost

Even for *1000s* of design variables

Design Variables in FUN3D

- Global flowfield variables
  - Mach number, angle of attack, sideslip, noninertial rates
- Shape variables
  - These depend entirely on the geometric parameterization being supplied to FUN3D
  - FUN3D has no native shape variables, other than the grid points themselves
- Additional variables related to unsteady simulations

Objective/Constraint Functions in FUN3D

$$f_i = \sum_{j=1}^{J_i} \omega_j (C_j - C_j^*)^{p_j}$$

ω = weight C = aero coeff
p = power C^* = target aero coeff

- We call each term in the summation a *component* function and the summation f_i a *composite* function
- User may specify which boundary patch in the grid (or all) to which each component function applies
- Constraints may be explicit or added as "penalties"
- Multipoint/multiobjective: as many composite functions/constraints as desired
  - Only limited by particular optimization package
  - Adjoints for multiple functions/constraints computed simultaneously
- The optimization always seeks to *minimize* the objective function(s), so pose them accordingly
- This general form leads to numerous ways to pose an optimization problem; each optimizer has its own limitations though
  - Extensive discussion in manual

Objective/Constraint Functions Examples

Unconstrained Drag Minimization

$$f = C_D^2$$

Drag Minimization with C_L=0.5 Lift Penalty

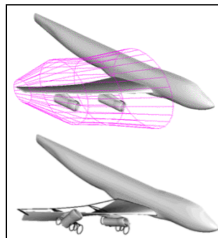
$$f = 10C_D^2 + (C_L - 0.5)^2$$

Drag Minimization with Explicit C_L=0.5 Lift Constraint

$$f_1 = C_D^2 \qquad f_2 = C_L$$

Geometry Parameterizations

- FUN3D relies on a predefined relationship between a set of parameters, or design variables, and the discrete surface mesh coordinates
- Given $\mathbf{D}$, surface parameterization determines $\mathbf{X}_{surf}$ (surface mesh)
- For example, given the current value of wing thickness at a location, what are the corresponding xyz-coordinates of the mesh?
- This narrows down the number of design variables from hundreds of thousands (raw grid points) to dozens or hundreds
  - Optimizers will perform more efficiently
  - Smoother design space
- The other requirement of the parameterization package is that it provides the Jacobian of the relationship between the design variables and the surface mesh, $\partial\mathbf{X}_{surf}/\partial\mathbf{D}$
- While users may provide their own parameterization scheme, FUN3D is set up to handle three common packages:
  - MASSOUD: Aircraft-centric design variables (thickness, camber, planform, twist, etc.)
  - Bandaids: General patching tool to handle fillets, winglets, and other odd shapes
  - Sculptor: Commercial package from Optimal Solutions
- To dump out the surface grids in the Tecplot format necessary for these tools, run the flow solver with '`--write_massoud_file`'
  - This procedure generates a `[project]_massoud_bndryN.dat` file for the i^th solid boundary



Wing Twist via MASSOUD

Directory Tree for FUN3D-Based Design

`Design`
- Main directory for design execution
- The only directory here without a hardwired name

`Design/ammo`
- Design is executed from here using the `opt_driver` executable
- `design.nml` resides here

`Design/description.i`
- `i` suffix is an integer referring to the design point (to accommodate multipoint design)
- Contains all of the baseline files describing this design point (CFD model and all input decks specific to it)
- **The optimization never changes anything in here**; this is where the optimizer can always find the problem definition
- **You provide the problem description for the `i`th design point here**

`Design/model.i`
- `i` suffix is an integer referring to the design point (to accommodate multipoint design)
- All CFD runs are performed here
- **You never change anything in here**; it only contains outputs

`Design/model.i/Flow`
- All flow solutions are performed here

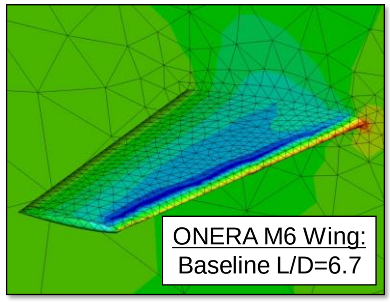
`Design/model.i/Adjoint`
- All adjoint solutions are performed here

`Design/model.i/Rubberize`
- All parameterization evaluations are performed here

`Design/model.i/Rubberize/surface_history`
- A Tecplot file for every surface grid evaluated during the design is stored here

You need not set up this tree manually; the code will do it for you, provided some basic pathnames

Maximize L/D for Transonic Flow Over a Wing



- To create the directory structure necessary for performing the optimization, issue the following command:

  `'/path/to/your/FUN3D/installation/Design/opt_driver --setup_design 1'`
- The trailing integer represents the number of design points desired
- This command will prompt you for several paths and then will set up the required directory structure
- First we will discuss the files that must be provided in the `description.1` directory

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

`command_line.options`

```
3
0 flow
1 adjoint
'--rmstol 1.e-3'
0 mpirun
```

- This file is used to specify any command line options (CLOs) required by the FUN3D executables, as well as MPI
- The first line specifies the number of executables for which you are providing CLOs
- This is followed by a line containing an integer and a keyword
  - The integer specifies the number of CLOs you are providing for the code identified by the keyword
- This is followed by the actual CLOs for the current executable
- Note 'mpirun' is an available keyword: this provides a mechanism to feed your `mpirun` executable any options it may require (`-nolocal`, `-machinefile filename`, etc.)
  - Depends on your environment, queue structure, etc.

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` ***Directory***

We are assuming the use of a MASSOUD parameterization for this example

`design.1, design.gp.1`

- These files are input files for MASSOUD for the 1st body; the MASSOUD setup tool provides these when you set up your parameterization
- Do not change these files

`design.usd.1`

- This file is an input file for MASSOUD for the 1st body; the MASSOUD setup tool provides this template when you set up your parameterization
- Depending on how you choose to “link” raw MASSOUD variables to create new variables, this defines the linking weights (see MASSOUD documentation)
- When using MASSOUD with FUN3D, you must always use the design variable linking option, even if simply set to the identity matrix

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` ***Directory***

`design.usd.1`

```
# this is input sd file for MASSOUD
# number of row == number dvs within MASSOUD
# number of col == final number  dvs
#(row) (col) (#of nonzero rows)
10 11 10
     d   1d   2d   3d   4d   5d   6d   7d   8d   9d  10d  11d
     1    1    0    0    0    0    0    0    0    0    0    0
     2    0    1    0    0    0    0    0    0    0    0    0
     3    0    0    1    0    0    0    0    0    0    0    0
     4    0    0    0    1    0    0    0    0    0    0    0
     5    0    0    0    0    1    0    0    0    0    0    0
     6    0    0    0    0    0    1    0    0    0    0    0
     7    0    0    0    0    0    0    1    0    0    0    1
     8    0    0    0    0    0    0    0    1    0    0    1
     9    0    0    0    0    0    0    0    0    1    0    1
    10    0    0    0    0    0    0    0    0    0    1    1
```

- Our demo problem uses 166 variables; this sample file only shows 10 raw variables plus 1 linked variable for clarity
- Linked variable is equal combination of raw DV’s 7-10

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

`massoud.1`

```
#MASSOUD INPUT FILE
# runOption (0 analysis), (> 0 sd using user's dvs ) (-1, sd using massoud's dvs)
166
# core (0 incore solution)(1 out of core solution)
0
# input parameterized file
design.gp.1
# design variable input file
design.1
# input sensitivity file (used for runOption > 0
design.usd.1
# output file grid file
new1.plt
# output tecplot file for viewing
model.tec.1
# file containing the design variables group
designVariableGroups.1
# user design variable file
customDV.1
```

- This file tells MASSOUD the names of its input/output files for the 1st body
- The first value specifies the number of linked MASSOUD design variables
  - If linking matrix is identity, this is just the number of raw MASSOUD design variables
- The remainder of the inputs are filenames; they should remain as is, but with the integer value in each name set to the index of the current body

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

`fun3d.nml`

- This is the nominal solver input deck for your case
- The adjoint solver also uses this input
  - If the adjoint requires different values (e.g., stopping tolerance), you can override these values with CLOs given in `command_line.options`
- It should contain the necessary inputs to run the baseline case
- The optimization will override values as needed using CLOs (e.g., angle of attack, etc.)

`[project].fgrid, [project].mapbc`

- This is the nominal mesh for your baseline case in whatever grid format is convenient

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

`rubber.data`

- This is the main design control file used to define the design variables and their bounds, objective functions, and constraints for the current design point
- It also stores current values of functions and sensitivities
- A copy of this file is placed in the `model.1` directory at the beginning of an optimization and is continuously updated with the current values of the design variables, objective/constraint functions, and all gradient information
  - If you want to know the latest info during a design, it's probably in here

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

`rubber.data`: Design Variable Block

- In general, for each design variable, you must set several fields
  - Active (0=no, 1=yes), baseline value, upper and lower bounds (if active)
- First subsection lays out global design variable information including Mach number, angle-of-attack, yaw, noninertial rates
- This is followed by an input stating the number of bodies to be designed
- Then for each body:
  - Fixed number of rigid motion variables – leave these alone (used for unsteady flows)
  - Number of shape variables and their inputs – these correspond directly to the MASSOUD variables previously discussed
    - When setting bounds for shape variables, it pays to be conservative – the optimizer will exploit every radical shape it can dream up
    - You can quickly get into unsolve-able or invalid/crossed-up geometries
    - You can always loosen up the bounds and restart the design if needed

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

```
################################################################################
######################## Design Variable Information ###########################
################################################################################
Global design variables (Mach number / angle of attack)
Index Active         Value                  Lower Bound            Upper Bound
 Mach    0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
  AOA    0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
  Yaw    0    0.000000000000000E+00  0.000000000000000E+00 0.000000000000000E+00
xrate    0    0.000000000000000E+00  0.000000000000000E+00 0.000000000000000E+00
yrate    0    0.000000000000000E+00  0.000000000000000E+00 0.000000000000000E+00
zrate    0    0.000000000000000E+00  0.000000000000000E+00 0.000000000000000E+00
Number of bodies
    1
Rigid motion design variables for body 1 (name of body 1, less than 80 cols)
  Var Active         Value                  Lower Bound            Upper Bound
RotRate  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotFreq  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotAmpl  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotOrgx  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotOrgy  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotOrgz  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotVecx  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotVecy  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
RotVecz  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
TrnRate  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
TrnFreq  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
TrnAmpl  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
TrnVecx  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
TrnVecy  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
TrnVecz  0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
Parameterization Scheme (Massoud=1 Bandaids=2 Sculptor=4)
1
Number of shape variables for body 1 (name of body 1, less than 80 cols)
  166
Index Active         Value                  Lower Bound            Upper Bound
    1    0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
    2    0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
    3    0    0.000000000000000E+00  0.000000000000000E+00 0.500000000000000E+01
.
.
```


Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

`rubber.data`: Function Block

- These sections lay out the objective/constraint function definitions
- First input is the total number of composite functions being specified (sum of objectives + constraints)
- Then, for each function:
  - Is it an objective function (1) or a constraint (2)
  - If it is a constraint, what are the upper and lower bounds (otherwise dummies)
  - How many component functions are used to build up the composite function
  - Time step interval defining the function (leave as dummies – for unsteady design)
  - Composite function weight/target/power: for further generality, described in manual
  - Then the list of component functions:
    - Boundary index it applies to (0 means all boundaries)
    - Keyword identifying the function type (see manual)
    - Value (dummy – this is an output during the optimization)
    - Weight/target/power to be applied to current component function
    - The remainder of the function block is devoted to sensitivity outputs – you can place dummies here, *but there must be a line corresponding to every design variable*

Maximize L/D for Transonic Flow Over a Wing

Files Required in `description.1` Directory

```
################################################################################
############################# Function Information #############################
################################################################################
Number of composite functions for design problem statement
    1
################################################################################
Cost function (1) or constraint (2)
    1
If constraint, lower and upper bounds
    0.0 0.0
Number of components for function    1
    1
Physical timestep interval where function is defined
    1 1
Composite function weight, target, and power
1.0 0.0 1.0
Components of function   1: boundary id (0=all)/name/value/weight/target/power
    0 clcd          0.000000000000000           1.000    20.00000 2.000
Current value of function    1
             0.000000000000000
Current derivatives of function wrt global design variables
             0.000000000000000
             0.000000000000000
.
.
.
Current derivatives of function wrt rigid motion design variables of body    1
             0.000000000000000
             0.000000000000000
.
.
.
Current derivatives of function wrt design variables of body    1
             0.000000000000000
             0.000000000000000
.
.
.
```

Our objective function:
$f = (L/D - 20)^2$

Maximize L/D for Transonic Flow Over a Wing

`ammo/design.nml`

- We are now finished setting things up in the `description.1` directory
- There is one more file that needs to be set up in the `../ammo` directory
- The `design.nml` file controls the actual optimization procedure
- Everything in this namelist file is pretty self-explanatory, but a few reminders:
  - '`opt_algorithm`': DOT/BIGDOT=1, KSOPT=3, PORT=4, NPSOL=5, SNOPT=6
  - '`what_to_do`': analysis=1, sensitivity analysis=2, optimization=3
  - Note you can specify the mpirun executable name
    - Useful if executable is called '`mpiexec`', '`aprun`', or otherwise on your system
  - Otherwise, see extensive documentation for this namelist in the manual

```
&design
  base_directory  =  'path/to/your/design/case'
  what_to_do      =  1
  mpirun_prefix   =  'mpiexec'
/
```

Maximize L/D for Transonic Flow Over a Wing

Running a Function Evaluation

- Things are now ready for execution
- The first thing I typically do is just run a function evaluation to see that the parameterization and all of the inputs are set correctly
- To do this, edit `design.nml` and set `what_to_do` to `1`
- From the `ammo` directory, the command line that is used to run this case is

```
./opt_driver --sleep_delay 5
```

  - The '`--sleep_delay 5`' instructs the design driver to wait 5 seconds in between operations – allows NFS caching to keep up
  - Different systems may require more time (or none)

Maximize L/D for Transonic Flow Over a Wing

Running a Function Evaluation

- The first thing that you will see is MASSOUD evaluating the parameterization for each body, defining the surface grid coordinates at the baseline position
- The flow solver will then start up, but prior to the solve, you will see an auxiliary solution take place that represents the interior mesh movement based on the elasticity equations
  - For this first step at the baseline position, you should see very small numbers for the "`Natural Error Est`" (close to machine zero): this indicates the current surface mesh is very close to the requested surface mesh
- After the actual flow solution takes place, the solver will evaluate each of the objective and constraint functions you posed:

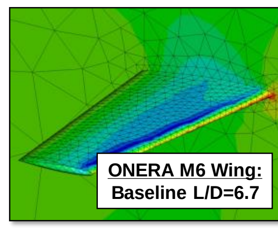
```
Current value of function      1       178.087727962997
```

- This marks the end of a successful function evaluation
- Always wise to plot the flow solver convergence – you want to run enough iterations to get a "reasonable" answer (outputs resolved beyond what you are expecting from design changes), but you don't necessarily need to drive it into the ground

Maximize L/D for Transonic Flow Over a Wing

Running a Function Evaluation

```
[MASSOUD Screen Output]

 Sleeping to allow file system time to catch up...

 Executing: mpiexec nodet_mpi --animation_freq -1 --design_run --irest 0 --write_mesh inviscid

FUN3D 12.7-74063 Flow started 05/20/2015 at 14:38:54 with 24 processes
  [Echo of fun3d.nml]
  [Usual preprocessing info]
 Using linear elasticity to reposition grid...

 reading ../rubber.data ...
 reading:../Rubberize/model.tec.1.sd1
 Iter      Natural Err Est         Error Estimate      Restarts
    0    0.648914658284637E-16   0.000000000000000E+00         0

 Iter            density_RMS  density_MAX   X-location   Y-location   Z-location
    1  0.725550147064997E-04  0.46595E-03  0.34893E-01  0.60683E-01  0.00000E+00
       Lift  0.657554528793843E-01         Drag  0.319926994134964E-01
…
   74  0.207836490870309E-09  0.82846E-08  0.22500E+01  0.45000E+01  0.65000E+01
       Lift  0.881383268442809E-01         Drag  0.132438291863532E-01

 Writing boundary output: inviscid_tec_boundary.dat
  Time step: 74, ntt: 74, Prior iterations: 0

 Writing inviscid.flow (version 11.8) lmpi_io 2
  inserting current history iterations 74
 Time for write: .0 s
 Current value of function            1   178.087727962997
 writing ../rubber.data ...
 global element counts below i4 limit, write as 'stream'
 wrote inviscid.b8.ugrid in     0.0000
 Done.
 Analysis complete.
```



Maximize L/D for Transonic Flow Over a Wing

Running a Gradient Evaluation

- Now lets test a sensitivity analysis
- Edit `design.nml` and set `what_to_do` to `2`
- Submit the job just as before
- The first thing that will take place is a function evaluation, just as before
- After the function evaluation takes place, MASSOUD will fire up again to evaluate the linearizations of the surface mesh coordinates with respect to the design variables
- FUN3D's adjoint solver will then start up:
  - You will see a solution taking place; this is the flowfield adjoint
  - Afterwards, you will see another solution occurring; this is the elasticity adjoint for the mesh
  - The final step is to update the `model.1/rubber.data` file with the sensitivity information
- This marks the end of a successful sensitivity analysis
- Again, it is wise to plot the convergence of the flowfield adjoint system
  - This convergence history is in the `model.1/Adjoint/[project]_hist.dat` file
  - In general, you want 2-3 orders of magnitude convergence; this is usually sufficient for reasonable sensitivity information

Maximize L/D for Transonic Flow Over a Wing

Running a Gradient Evaluation

```
[Function Evaluation]

[MASSOUD Screen Output]

Sleeping to allow file system time to catch up...

Executing: mpiexec dual_mpi --rmstol 1.e-3 --getgrad --irest 0 --force_stream_file
FUN3D 12.7-74063 Adjoint started 05/20/2015 at 14:44:00 with 24 processes
 [Echo of fun3d.nml]
 [Usual preprocessing info]

 Iter          adjoint RMS    adjoint MAX    X location   Y location   Z location
    1  0.707037901636711E+00  0.30235E+01  0.57720E+00  0.95000E+00  0.13288E-01
    2  0.221413741319278E+02  0.77671E+03  0.22500E+01  0.45000E+01  0.65000E+01
    3  0.252132505507981E+02  0.85665E+03  0.22500E+01  0.45000E+01  0.65000E+01
…
   79  0.108404219416308E-02  0.48685E-01  0.20671E+00  0.43560E+01  0.19196E+01
   80  0.961305851711102E-03  0.43086E-01  0.20671E+00  0.43560E+01  0.19196E+01

 Performing linear elasticity adjoint...

 reading ../rubber.data ...
  Using defaults for move_relaxation.schedule.
 Boundary 1 allowed to deform with y=constant constraint
 Iter      Natural Err Est         Error Estimate       Restarts
    0    0.540562915758561E+04   0.100000000000000E+01         0
    1    0.351062487957891E+02   0.649438719756149E-02         0
   11    0.426070657988252E-02   0.788198090485649E-06         0
 writing ../rubber.data ...
 Done.
 Sensitivity analysis complete.
```


Maximize L/D for Transonic Flow Over a Wing

Running the Optimization

- If you got this far, things are looking pretty good – we've checked that everything is set up to run functions and gradients correctly, which is all the optimizer depends on
- Now we're ready to try an actual optimization
  - Edit `design.nml` and set `what_to_do` to `3`; submit the job like usual
- Now you will see a lot of function and gradient evaluations going by, as the optimizer starts to change design variables and search for an optimum solution
- One easy way to monitor progress is to grep your screen output:
  - **'grep "Current value" screen.output'**:

```
Current value of function         1   178.087727962997
Current value of function         1   137.781363854615
Current value of function         1   109.428434387371
Current value of function         1   95.6295324769749
Current value of function         1   98.1556907116245
Current value of function         1   90.6778940684516
Current value of function         1   90.5396512437177
Current value of function         1   87.6654699895390
Current value of function         1   87.6871503037963
Current value of function         1   87.1318763195701
Current value of function         1   86.8957999910668
Current value of function         1   87.3525539085617
Current value of function         1   86.5144811775675
Current value of function         1   86.8116026938974
Current value of function         1   86.2791203108911
Current value of function         1   86.2399423689607
Current value of function         1   86.2399415584093
```

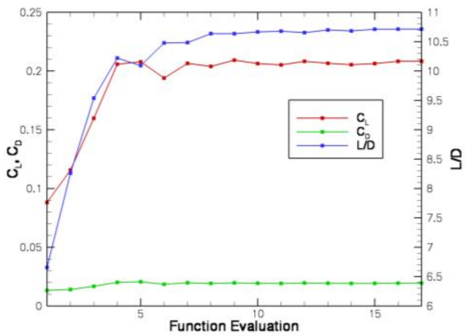
- You can also observe (but don't change!) the file `model.1/rubber.data`

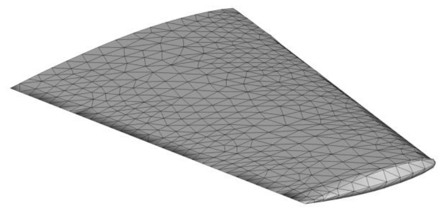
Maximize L/D for Transonic Flow Over a Wing

Post Mortem

- After the job finishes, PORT will summarize its performance in the file `model.1/port.output`
- Since each solution is a warm start, you can plot the entire flow solution history contained in `model.1/Flow/[project]_hist.dat`
- A history of the surface geometry is stored in `model.1/Rubberize/surface_history/model.tec.1.sd1.iteration.*`

Redesigned Wing:
L/D=10.7





What Could Possibly Go Wrong?

- The procedure can terminate due to CFD-related problems:
  - Running into negative volumes during a mesh movement (you can plot the history of the surface(s) using the files in `model.1/Rubberize/surface_history`)
    - Watch for invalid surfaces or unusually large changes
    - Be conservative in your lower/upper bounds!
  - The flowfield or the adjoint solution is unstable
    - Problem-dependent; get in touch for advice
- The procedure can also terminate due to hardware/environment problems
  - You run out of allocated time, a node dies, etc.
- Finally, the procedure can terminate if the optimizer has given up:
  - No more progress can be made due to constraints
  - The optimizer has hit the max number of functions/gradients you allowed
  - An optimal solution has been found

List of Key Input/Output Files

Input

- In `description.i` directory:
  - All files necessary to run solutions for `i`th design point (grid files, `fun3d.nml`, etc)
  - All parameterization files for `i`th parameterized body
  - `command_line.options`
  - `rubber.data`
- `ammo/design.nml`

Output

- All files normally associated with running the solver
- `rubber.data`
- `port.output`
- Design history in `model.1/Rubberize/surface_history`

Summary of Design Optimization for Steady Flows

- That's more or less the basic pieces involved with running an optimization
- Lots of options we did not cover here; see manual or get in touch for help
  - How the wrappers work (`LibF90/analysis.f90`, `LibF90/sensitivity.f90`)
  - Parameterizations other than MASSOUD
  - Multipoint/multiobjective (tutorial on website)
  - Constrained problems (tutorial on website)
  - Running with other optimization packages (tutorial on website)
  - Body grouping, spatial transforms
  - Archiving files during optimization
  - Overset grids
  - Forward-mode sensitivity analysis using complex variables
  - Unsteady design (later session)

General Advice

- Become very comfortable with the flow solver
- Work the tutorials
- Learn how to set up parameterizations using MASSOUD and/or bandaids
- Try plugging in your own grids/parameterizations in the tutorials
- Ask questions – it's actually not that bad once you get up the learning curve

What We Learned

- General approach used by FUN3D for design optimization
- What is an adjoint
- What does a function/gradient evaluation consist of in terms of CFD
- Design variables in FUN3D
- Functions/constraints in FUN3D
- What is required of a geometry parameterization tool
- How to set up the inputs required for design optimization
- How to run function, gradient evaluations
- How to perform a basic design optimization
- What to watch out for and how to interpret results