

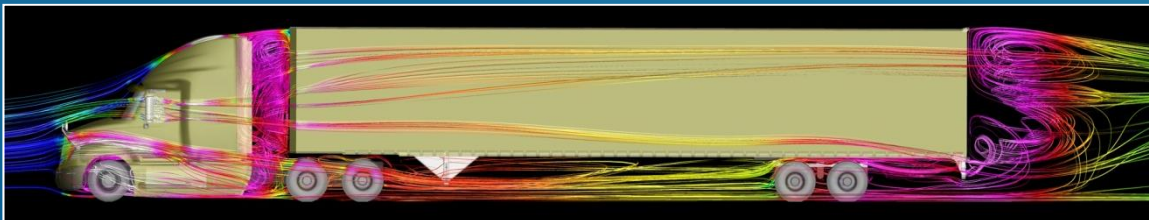
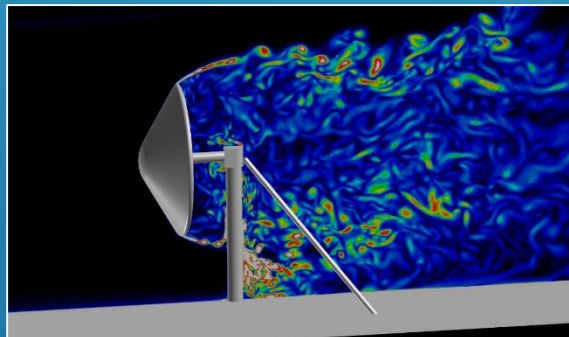
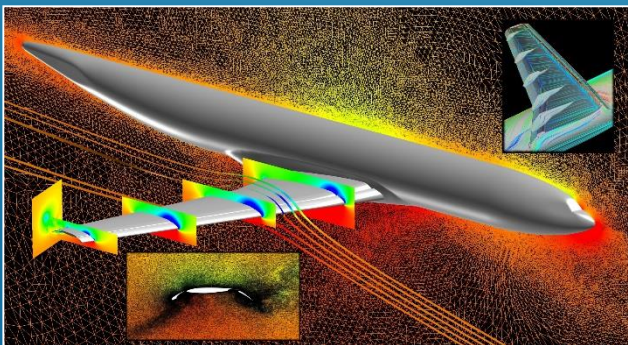
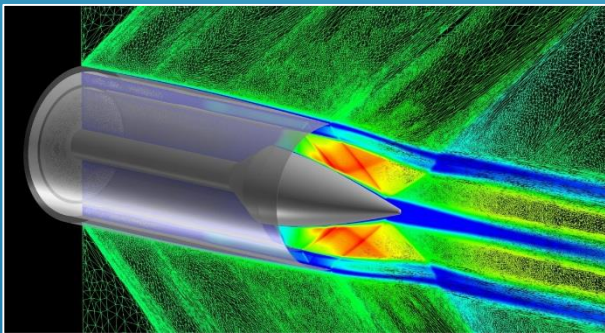
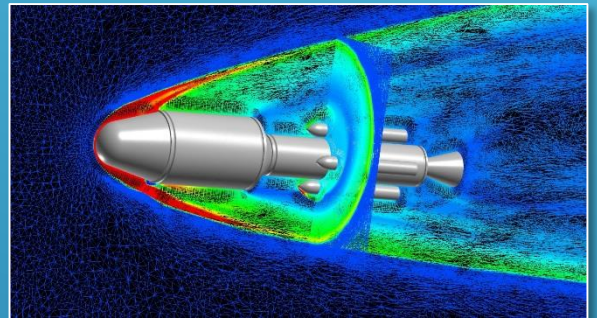
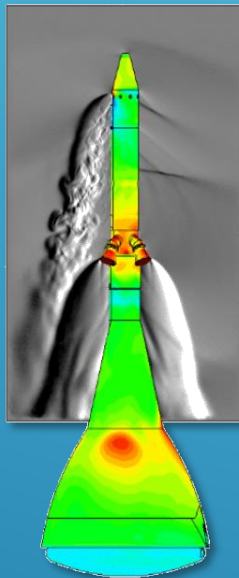
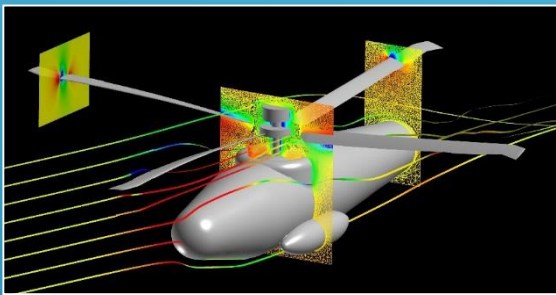
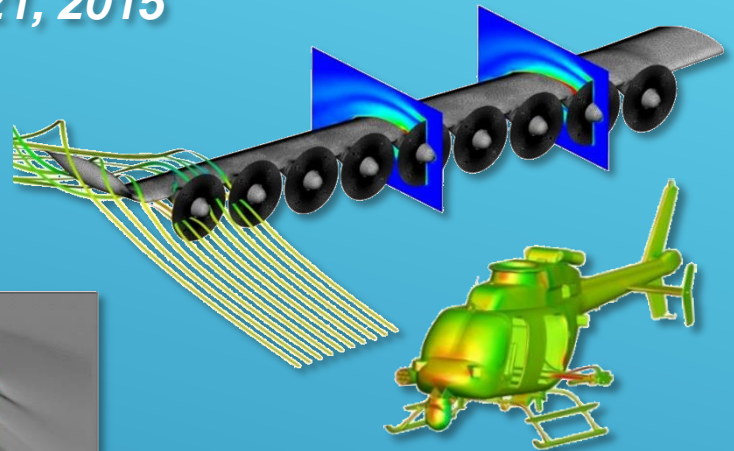
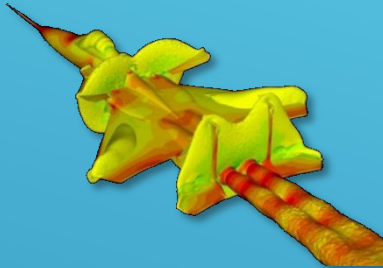


FUN3D

Fully Unstructured Navier-Stokes

Training Workshop

Dallas, Texas
June 20-21, 2015



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

FUN3D Training Workshop

June 20-21, 2015

Saturday, June 20

Session 1: Meet and Greet	All	8:00-8:30
Session 2 Welcome and Overview	Eric Nielsen	8:30-9:00
Session 3: Compilation and Installation	Bill Jones	9:00-9:15
Session 4: Griding, Solution, and Visualization Basics	Eric Nielsen	9:15-10:15
BREAK		10:15-10:30
Session 5: Boundary Conditions	Jan-Renee Carlson	10:30-11:00
Session 6: Turbulence Models	Jan-Renee Carlson	11:00-11:30
Session 7: Supersonic / Hypersonic Perfect Gas Simulations	Mike Park	11:30-12:00
CATERED LUNCH: Lightning Talks	Various	12:00-1:15
Session 8: Parameterization Tools	Bill Jones	1:15-2:15
Session 9: Adjoint-Based Design for Steady Flows	Eric Nielsen	2:15-3:45
BREAK		3:45-4:00
Session 10: Feature and Adjoint-Based Error Estimation and Mesh Adaptation	Mike Park	4:00-5:00



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

FUN3D Training Workshop

June 20-21, 2015

Sunday, June 21

Session 11: Time-Dependent Simulations	Bob Biedron	8:00-8:30
Session 12: Dynamic Grid Simulations	Bob Biedron	8:30-9:00
Session 13: Suggar ++	Ralph Noack	9:00-10:00
BREAK		10:00-10:15
Session 14: Overset Grid Simulations	Bob Biedron	10:15-10:45
Session 15: Adjoint-Based Design for Unsteady Flows	Eric Nielsen	10:45-12:00
LUNCH ON YOUR OWN		12:00-1:00
Session 16: Aeroelastic Simulations	Bob Biedron	1:00-1:45
Session 17: Rotorcraft Simulations	Bob Biedron	1:45-2:45
BREAK		2:45-3:00
Session 18: Current Development Activities, Summary of User Feedback and Requests	All	3:00-4:00
Session 19: High-Energy / Generic Gas Simulations *** Please see important note for this session below ***	Peter Gnoffo	4:00-4:30

Due to security regulations, workshop participants who would like to attend this session will be required to present a valid US passport as proof of US citizenship. There will be no exceptions to this requirement. The FUN3D team apologizes for any inconvenience this may cause.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

FUN3D Training – Day One Lunch Order

Your name:

Special notes (omit toppings, etc):

Total: \$18.00

Includes tax and gratuity

Exact change appreciated, but change is available

**** Circle one sandwich, one chips/cookie, and one drink ****

Chicken salad on croissant: Diced chicken, apples, grapes, mayonnaise

Turkey club: Smoked turkey, bacon, onion roll

Tuna salad: Tuna salad, green onion, wheat bread

Grilled vegetable wrap: Red bell peppers, yellow squash, mushrooms,
mixed greens, olive oil, spinach tortilla

Mrs. Vickie's Original Sea Salt Chips

Doritos Nacho Cheese Chips

Mrs. Vickie's Salt & Vinegar Chips

Chocolate chip cookie

Mrs. Vickie's Jalapeno Chips

Macadamia cookie

Kettle Cooked Original Sea Salt &
Cracked Pepper Chips

Oatmeal cookie

Doritos Cool Ranch Chips

Coke

Diet Dr. Pepper

Apple Juice

Diet Coke

Pink Lemonade

Grapefruit Juice

Sprite

Lemonade

Cranberry Juice

Coke Zero

Crystal Geyser Water

Dr. Pepper

Orange Juice

FUN3D User Feedback and Requests

The training team will summarize and discuss content received from all participants during the final session on Sunday. This is an opportunity to identify directions of mutual interest between NASA and your organization, and to gauge the level of interest across the user base. Please indicate any content below you wish to remain confidential; such items will be excluded from the public discussion on Sunday. Thanks!

Feedback: Are we doing something poorly? Suggestions for improvements?

Requests: What capabilities would you like to see in future versions of FUN3D?

Your name (optional):

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.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
2. I will be able to apply the knowledge learned.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
3. The training objectives for each topic were identified and followed.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
4. The content was organized and easy to follow.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
5. The materials distributed were pertinent and useful.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
6. The trainers were knowledgeable.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
7. The quality of instruction was good.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
8. The trainers met the training objectives.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
9. Class participation and interaction were encouraged.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
10. Adequate time was provided for questions and discussion.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

11. How do you rate the training overall?

Excellent	Good	Average	Poor	Very poor
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

12. What aspects of the training could be improved?

13. Other comments?

Your name (optional):

THANK YOU FOR YOUR PARTICIPATION!

FUN3D v12.7 Training

Session 1: Meet and Greet



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

Please Fill Out Your Saturday Lunch Request ASAP

- Lunch will be brought in for us today (on your own tomorrow)
- Brief “lightning” talks related to FUN3D while we eat
- Please circle one sandwich, one chips/cookie, and one drink
 - Note any special needs
- Total is \$18, including tax/gratuity – cash only
 - Exact change appreciated, but small change is available
- FUN3D team members will circulate shortly to collect orders and payments



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

Who is Here From the NASA Team?

All from NASA Langley in Hampton, Virginia:

Kyle Anderson, Computational AeroSciences Branch (observing)

Bob Biedron, Computational AeroSciences Branch

Jan-Renee Carlson, Computational AeroSciences Branch

Peter Gnoffo, Aerothermodynamics Branch

Bill Jones, Computational AeroSciences Branch

Eric Nielsen, Computational AeroSciences Branch

Mike Park, Computational AeroSciences Branch

Also:

Ralph Noack, Celeritas Simulation Technology, LLC



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

Who Are You?

Name

Organization/Company and Location

Any CFD Experience?

Any FUN3D Experience?

Goals/Uses for FUN3D



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

FUN3D v12.7 Training

Session 2: Welcome and Overview

Eric Nielsen



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

FUN3D Training Workshop

June 20-21, 2015

Saturday, June 20

Session 1: Meet and Greet	All	8:00-8:30
Session 2 Welcome and Overview	Eric Nielsen	8:30-9:00
Session 3: Compilation and Installation	Bill Jones	9:00-9:15
Session 4: Gridding, Solution, and Visualization Basics	Eric Nielsen	9:15-10:15
BREAK		10:15-10:30
Session 5: Boundary Conditions	Jan-Renee Carlson	10:30-11:00
Session 6: Turbulence Models	Jan-Renee Carlson	11:00-11:30
Session 7: Supersonic / Hypersonic Perfect Gas Simulations	Mike Park	11:30-12:00
CATERED LUNCH: Lightning Talks	Various	12:00-1:15
Session 8: Parameterization Tools	Bill Jones	1:15-2:15
Session 9: Adjoint-Based Design for Steady Flows	Eric Nielsen	2:15-3:45
BREAK		3:45-4:00
Session 10: Feature and Adjoint-Based Error Estimation and Mesh Adaptation	Mike Park	4:00-5:00



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

FUN3D Training Workshop

June 20-21, 2015

Sunday, June 21

Session 11: Time-Dependent Simulations	Bob Biedron	8:00-8:30
Session 12: Dynamic Grid Simulations	Bob Biedron	8:30-9:00
Session 13: Suggar ++	Ralph Noack	9:00-10:00
BREAK		10:00-10:15
Session 14: Overset Grid Simulations	Bob Biedron	10:15-10:45
Session 15: Adjoint-Based Design for Unsteady Flows	Eric Nielsen	10:45-12:00
LUNCH ON YOUR OWN		12:00-1:00
Session 16: Aeroelastic Simulations	Bob Biedron	1:00-1:45
Session 17: Rotorcraft Simulations	Bob Biedron	1:45-2:45
BREAK		2:45-3:00
Session 18: Current Development Activities, Summary of User Feedback and Requests	All	3:00-4:00
Session 19: High-Energy / Generic Gas Simulations *** Please see important note for this session below ***	Peter Gnoffo	4:00-4:30

Due to security regulations, workshop participants who would like to attend this session will be required to present a valid US passport as proof of US citizenship. There will be no exceptions to this requirement. The FUN3D team apologizes for any inconvenience this may cause.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

Administrative Details

- Need to stay on schedule, but please do not hesitate to ask questions
- Please submit your two forms by lunchtime on Sunday to any team member
 - **User Feedback/Requests Form**
 - User feedback and requests will be summarized and discussed in the final session on Sunday
 - **Training Evaluation Form**
 - Very interested in your feedback, good or bad!



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

All Material Available Online

- For the v12.7 material presented here:
 - Slides online in PDF format
 - Demo content can be downloaded as a tarball
 - Capture hopefully online soon
- A FUN3D v12.7 manual is available as NASA/TM-2015-218761 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
- We hope to eventually add an extensive tutorials document



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

The FUN3D Support Team

fun3d-support@lists.nasa.gov

“Who sees my questions to the support alias?”

- Consists of 14 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, Peter Gnoffo, Dana Hammond, Bill Jones, Bil Kleb, Beth Lee-Rausch, Steve Massey, Eric Nielsen, Matt O’Connell, Mike Park, Kyle Thompson, 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.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

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 100 distributions / year
 - Wide range of uses including aerospace, automotive, HPC, 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>

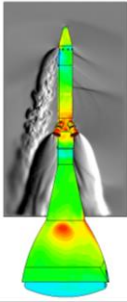
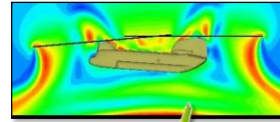
FUN3D Training Workshop
June 20-21, 2015



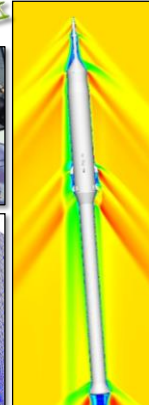
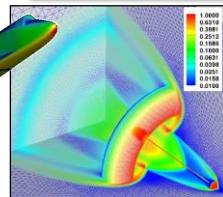
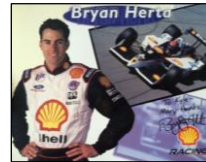
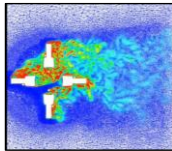
8

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



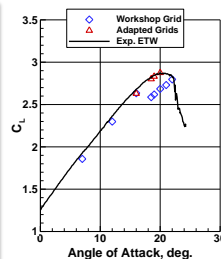
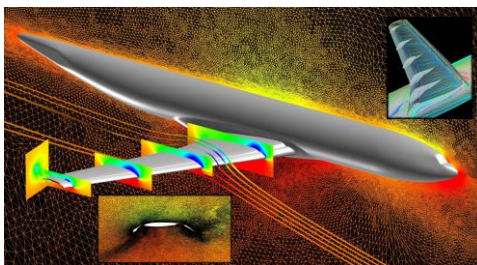
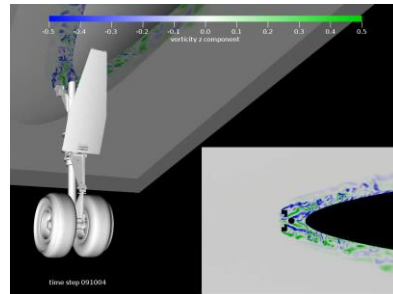
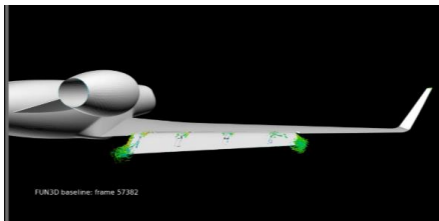
Georgia
Tech



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

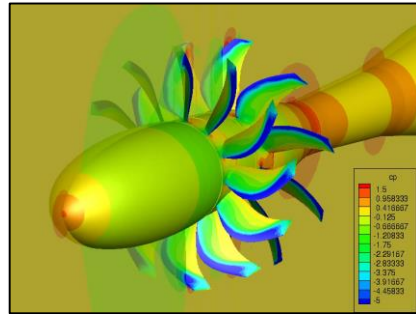


Courtesy
Bob Bartels



**Aeroelastic Analysis of
the Boeing SUGAR
Truss-Braced Wing
Concept**

Open-Rotor Concepts



Courtesy Bill Jones



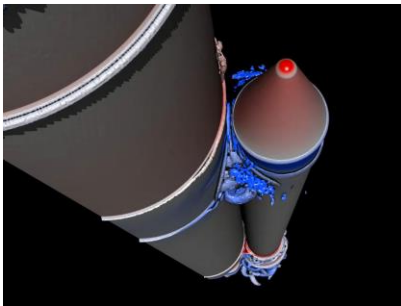
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



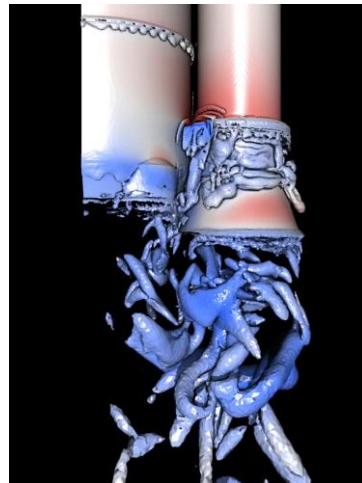
11

Some Recent NASA Applications



**Transonic Buffet
Characterization for
Space Launch System**

Courtesy
Greg Brauckmann,
Steve Alter, Bill Kleb



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

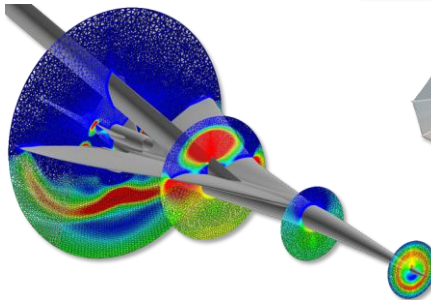
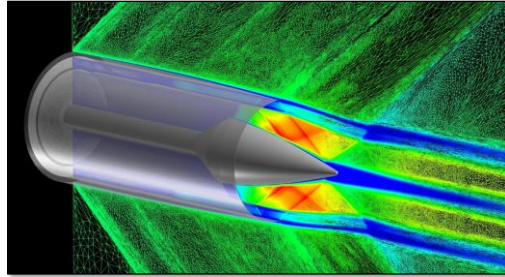


12

Some Recent NASA Applications

Courtesy
Chris Heath

Sonic Boom Mitigation



Mars InSight Lander



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

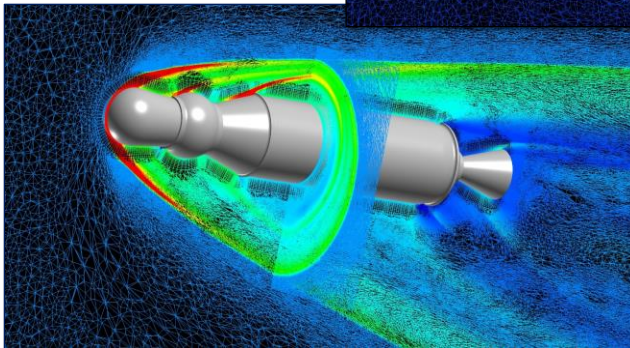
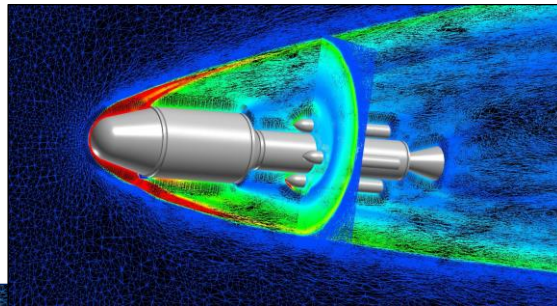


13

Some Recent NASA Applications

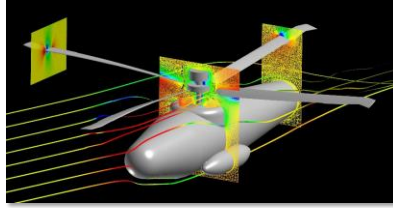
**Mars Ascent Vehicle
for Sample Return**

Courtesy
Ashley Korzun



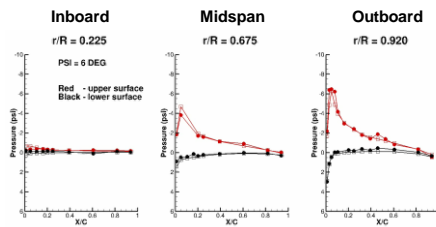
Some Recent NASA Applications

Validation for Full Scale UH60A

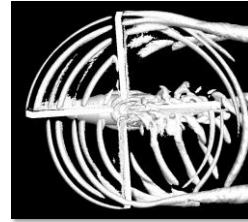
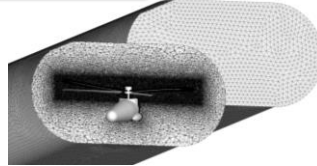


Courtesy
Beth Lee-Rausch,
Bob Biedron

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

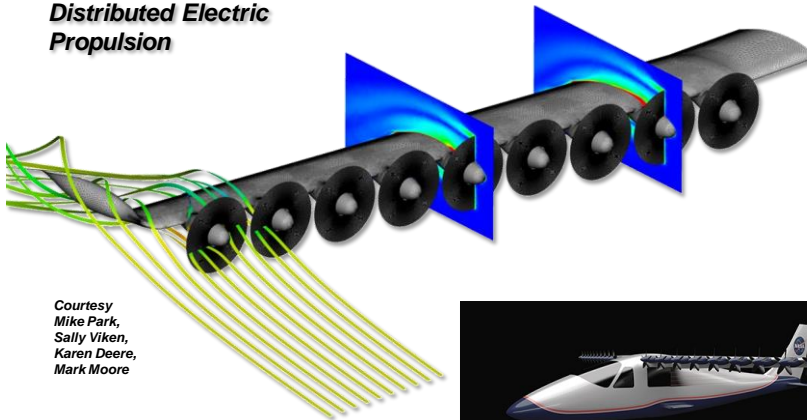


Blade Pressures at High Advance Ratio



Some Recent NASA Applications

Distributed Electric Propulsion



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



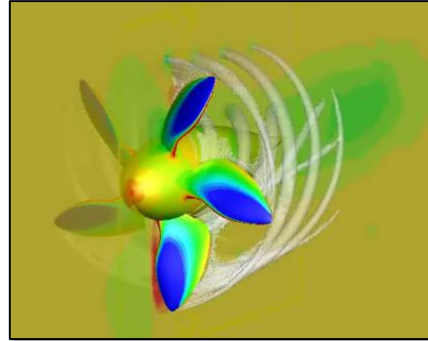
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

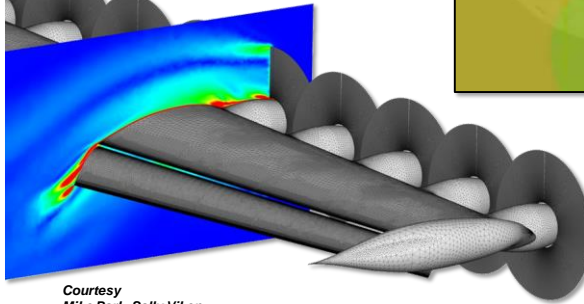


16

Some Recent NASA Applications



Courtesy Bill Jones



Distributed Electric Propulsion

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



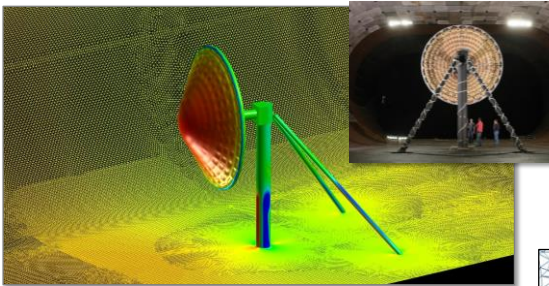
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



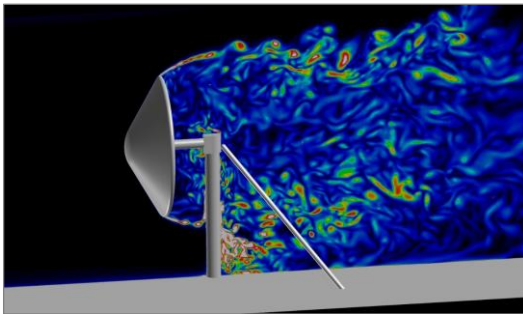
17

Some Recent NASA Applications

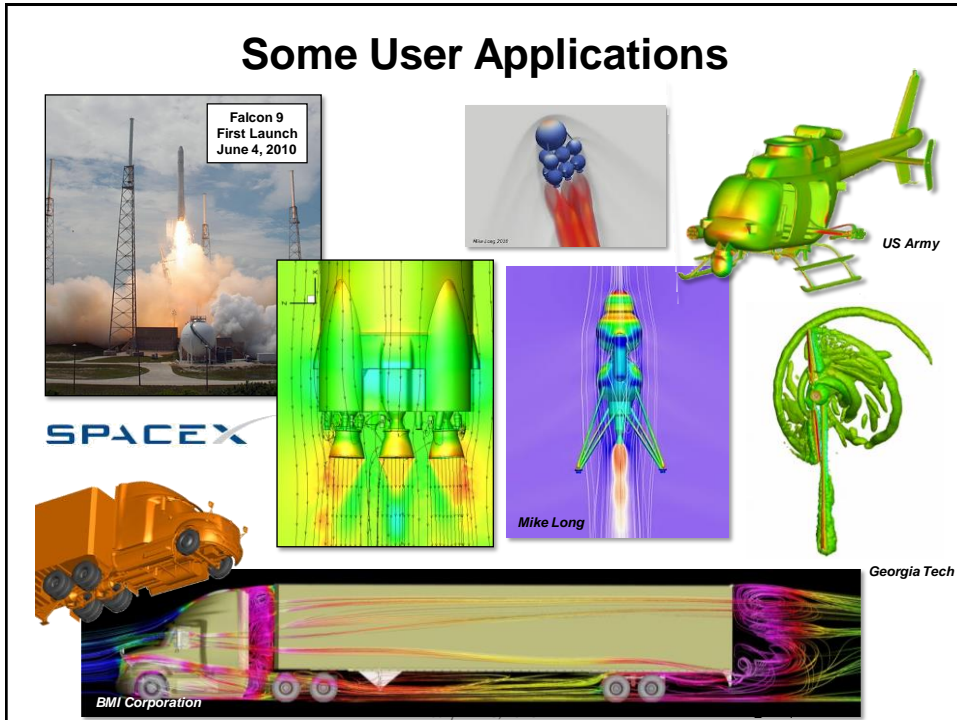


**Aeroelastic Analysis of
HIADs: Hypersonic
Inflatable Aerodynamic
Decelerators**

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

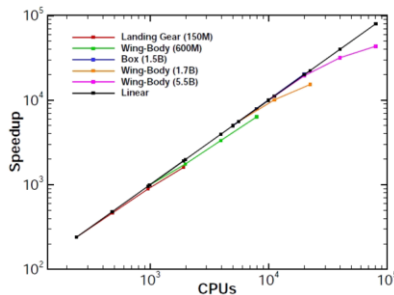


Some User Applications



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
June 20-21, 2015



20

Some Final Notes

- The material that will be shown here represents the current recommended best practices for the perfect gas option in FUN3D
- Simulations with real gas effects are covered Sunday afternoon for users who present a valid US passport
- 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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



21

FUN3D v12.7 Training

Session 3:

Compilation and Installation

Bill Jones



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

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 **NOT** 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 **NOT** be edited by hand
 - All localization is managed through the `configure` script
 - Checks various details required by compilation
 - Fails fast (prior to compilation of FUN3D) if problems are detected with the configuration options (no compiler, missing libraries, etc.)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

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`]



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

Nuts and Bolts (3 of 4)

- FUN3D optional Packages of general interest

<code>--with-PACKAGE [=ARG]</code>	use PACKAGE [<code>ARG=yes</code>]	
<code>--without-PACKAGE</code>	do not use PACKAGE (same as <code>--with-PACKAGE=no</code>)	
<code>--with-mpi [=ARG]</code>	Path to MPI library	(installation root)
<code>--with-mpibin [=ARG]</code>	MPI binary directory	(relative, absolute, without)
<code>--with-mpif90 [=ARG]</code>	MPI Fortran compiler wrapper	(relative, absolute, without)
<code>--with-mpicc [=ARG]</code>	MPI C compiler wrapper	(relative, absolute, without)
<code>--with-mpiexec [=ARG]</code>	MPI execution startup script	(relative, absolute, without)
<code>--with-mpibin [=ARG]</code>	MPI bin directory	(relative, absolute, without)
<code>--with-mpiinc [=ARG]</code>	Path to "mpif.h"	(relative, absolute, without)
<code>--with-parmetis [=ARG]</code>	ParMetis install path	(contains lib/libparmetis.a)
<code>--with-dirtlib [=ARG]</code>	use DiRTLib overset library	(contains lib/libdirt.a)
<code>--with-suggar [=ARG]</code>	use SUGGAR overset library	(contains lib/lib suggar.a)
<code>--with-tecio [=ARG]</code>	Tecplot I/O library install path	(contains lib/libtecio.a)
<code>--with-refine [=ARG]</code>	use refine adaptation package	(installation root)
<code>--with-refineFAKEGeom [=ARG]</code>	to specify refine FAKEGeom libs [<code>-lFAUXGeom</code>]	
<code>--with-knife [=ARG]</code>	use Knife cut cell package	(installation root)
<code>--with-CGNS [=ARG]</code>	CGNS library path	(installation root)
<code>--with-PORT [=ARG]</code>	use PORT optimization library	(contains lib/libport.a)
<code>--with-KSOPT [=ARG]</code>	use KSOPT optimization library	(contains lib/libksopt.a)
<code>--with-SNOPT [=ARG]</code>	use SNOPT optimization library	(contains lib/libsnopt.a)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

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`)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

Did It Work? Expected Output

...		knife:	subpackage
Configuration (FUN3D):		MPI support:	no
Source code location: ..		CUDA support:	no
Version: 12.7-74063		Zoltan:	no
Fortran Compiler: ifort		ParMETIS:	no
Fortran basis: ifort		Tecplot I/O:	no
Fortran flags: -O2 -ip -align		6DOF libraries:	no
-fno-alias -g -traceback		DiRTlib support:	no
C Compiler: gcc		SUGGAR support:	no
C flags: -g -O2		DYMORE support:	no
C++ Compiler: g++		RCAS_SDx support:	no
C++ flags: -g -O2		CGNS support:	no
Linker flags: -lm		PORT support:	no
Dependencies:		NPSOL support:	no
build:		DOT support:	no
High Energy Physics: no		KSOPT support:	no
Cmplx Variable Tools: no		SNOPT support:	no
Python bindings: no		SMEMRD support:	version 1.3.1
FCCHT support: no		IRS support:	no
FSI support: no		SSDC support:	no
PDF documentation: yes		SFE support:	no
		SPARSKIT support:	no
		SBOOM support:	no
		VisIt support:	no
bindings:			
Libcore: internal			
refine: subpackage			
CAPRI support: no	page 1		page 2

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



9

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
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

Did It Work? Expected Output

...		knife:	subpackage
Configuration (FUN3D):		MPI support:	no
Source code location: ..		CUDA support:	no
Version: 12.7-74063		Zoltan:	no
Fortran Compiler: /path/to/mpi/bin/mpif90		ParMETIS:	/path/to/parmetis
Fortran basis: ifort		Tecplot I/O:	no
Fortran flags: -O2 -ip -align		6DOF libraries:	no
-fno-alias -g -traceback		DiRTlib support:	no
C Compiler: /path/to/mpi/bin/mpicc		SUGGAR support:	no
C flags: -g -O2		DYMORE support:	no
C++ Compiler: g++		RCAS_SDx support:	no
C++ flags: -g -O2		CGNS support:	no
Linker flags: -lm		PORT support:	no
Dependencies:		NPSOL support:	no
build:		DOT support:	no
High Energy Physics: no		KSOPT support:	no
Cmplx Variable Tools: no		SNOPT support:	no
Python bindings: no		SMEMRD support:	version 1.3.1
FCCHT support: no		IRS support:	no
FSI support: no		SSDC support:	no
PDF documentation: yes		SFE support:	no
		SPARSKIT support:	no
		SBOOM support:	no
		VisIt support:	no
bindings:			
Libcore: internal			
refine: subpackage			
CAPRI support: no	page 1		page 2

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

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)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



12

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``



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

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_mpf90
               --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
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

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 v12.7 Training

Session 4: Gridding, Solution, and Visualization Basics

Eric Nielsen



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

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`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

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/ft³)

– μ_{ref}^* = reference molecular viscosity (e.g. slug/ft-s)

– T_{ref}^* = reference temperature (e.g. °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:

$$-\vec{x} = \vec{x}^* / (L_{ref}^* / L_{ref}) \quad t = t^* a_{ref}^* / (L_{ref}^* / L_{ref}) \quad t = t^* U_{ref}^* / (L_{ref}^* / L_{ref})$$

(compressible) (incompressible)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

Nondimensionalization (cont)

- For the **compressible flow** equations the dimensionless variables are:

$$-\vec{u} = \vec{u}^* / a_{ref}^* \quad \text{so } |\vec{u}|_{ref} = |\vec{u}|_{ref}^* / a_{ref}^* = M_{ref}$$

$$-P = P^* / (\rho_{ref}^* a_{ref}^{*2}) \quad \text{so } P_{ref} = P_{ref}^* / (\rho_{ref}^* a_{ref}^{*2}) = 1/\gamma$$

$$-a = a^* / a_{ref}^* \quad \text{so } a_{ref} = 1$$

$$-T = T^* / T_{ref}^* \quad \text{so } T_{ref} = 1$$

$$-e = e^* / (\rho_{ref}^* a_{ref}^{*2}) \quad \text{so } e_{ref} = e_{ref}^* / (\rho_{ref}^* a_{ref}^{*2}) = 1/(\gamma(\gamma-1)) + M_{ref}^2/2$$

$$-\rho = \rho^* / \rho_{ref}^* \quad \text{so } \rho_{ref} = 1$$

- 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

$$\text{reynolds_number} = \text{Re}_{ref} / L_{ref} \quad \text{where } \text{Re}_{ref} = \rho_{ref}^* U_{ref}^* L_{ref}^* / \mu_{ref}^*$$

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

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



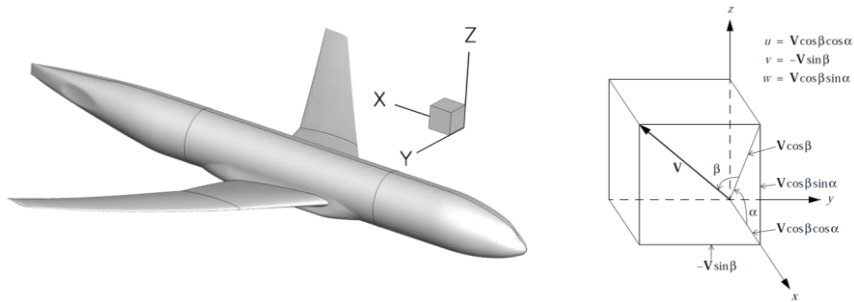
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

FUN3D Axis Convention



- FUN3D coordinate system differs from the standard wind coordinate system by a 180° 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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



9

Runtime Environment

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


```
$ ulimit unlimited # for bash
$ ulimit          # 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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

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



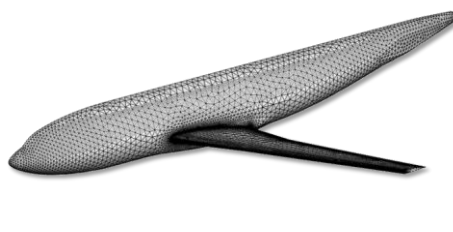
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



12

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	surfIDe	Family
1	3	3	0	0	0	Box
2	3	3	0	0	0	Box
3	3	3	0	0	0	Box
4	3	3	0	0	0	Box
5	3	3	0	0	0	Box
6	4	4	1	15	15	Wing
7	4	4	1	15	15	Wing
8	4	4	1	17	17	Wing
9	4	4	1	17	17	Wing
10	4	4	1	15	15	Wing
11	4	4	1	13	13	PoseLage
12	4	4	1	21	21	PoseLage
13	4	4	1	11	11	PoseLage
14	4	4	1	11	11	PoseLage
15	4	4	1	12	12	PoseLage
16	4	4	1	12	12	PoseLage
17	4	4	1	15	15	Wing
18	4	4	1	15	15	Wing
19	4	4	1	15	15	Wing
20	4	4	1	15	15	Wing
21	4	4	1	17	17	Wing
22	4	4	1	17	17	Wing
23	4	4	1	15	15	Wing
24	4	4	1	15	15	Wing
25	4	4	1	17	17	Wing
26	4	4	1	8	8	PoseLage
27	4	4	1	16	16	Wing
28	4	4	1	16	16	Wing
29	4	4	1	16	16	Wing
30	4	4	1	16	16	Wing
31	4	4	1	18	18	Wing
32	4	4	1	18	18	Wing
33	4	4	1	18	18	Wing
34	4	4	1	18	18	Wing
35	4	4	1	18	18	Wing
36	4	4	1	1	1	Wing
37	4	4	1	18	18	Wing
38	4	4	1	18	18	Wing
39	4	4	1	18	18	Wing
40	4	4	1	22	22	PoseLage
41	1	1	0	0	0	Symmetry
42	4	4	1	10	10	PoseLage
43	4	4	1	9	9	PoseLage
44	4	4	1	14	14	PoseLage
45	4	4	1	23	23	PoseLage
46	4	4	1	15	15	Wing
47	4	4	1	20	20	Wing
48	4	4	1	27	27	Fairing
49	4	4	1	29	29	Fairing
50	4	4	1	28	28	Fairing
51	4	4	1	30	30	Fairing



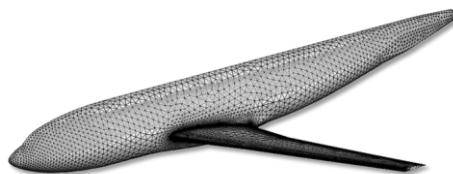
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

Transonic Turbulent Flow on a Tetrahedral Wing-Body Mesh

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



15

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



16

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,nnodes 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
```

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

```
... FM (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
```

of edges cut by partitioning (measure of communication)

1.6 secs required to preprocess the mesh

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

Min/max/avg wall spacing statistics



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



17

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	0.567457404772944E+00	0.28035E+02	0.16377E+03	-0.16562E+03	0.20117E+02
	0.764159584901413E+04	0.13249E+07	0.79654E+04	-0.88280E+04	0.25675E+02
	LiFt 0.103226565173772E+00		Drag 0.646513396068887E+00		
2	0.300679598726331E+00	0.12718E+02	0.29226E+03	-0.72487E+02	-0.12411E+02
	0.753354470463467E+04	0.12868E+07	0.79654E+04	-0.88280E+04	0.25675E+02
	LiFt 0.146829230859457E+00		Drag 0.721243167013704E+00		
.					
.					
.					
999	0.383370843514542E-05	0.13909E-03	0.35380E+03	-0.58429E+02	-0.16200E+02
	0.318320572426105E-02	0.19891E+00	0.36848E+03	-0.68458E+02	0.31074E+01
	LiFt 0.556387990643583E+00		Drag 0.388233647462313E-01		
1000	0.382497896407724E-05	0.13871E-03	0.35380E+03	-0.58429E+02	-0.16200E+02
	0.317436044959994E-02	0.19835E+00	0.36848E+03	-0.68458E+02	0.31074E+01
	LiFt 0.556387923023456E+00		Drag 0.388233658091165E-01		

Writing f6fx2b.flow (version 11.8) lmpi_io 2

inserting current history iterations 1000

Time for write: .1 s

Done.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



18

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.

```



<http://fun3d.larc.nasa.gov>

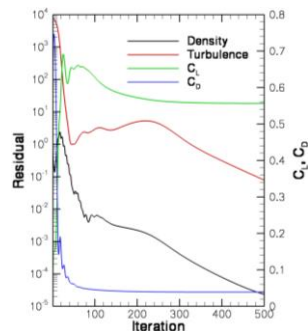
FUN3D Training Workshop
June 20-21, 2015



19

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



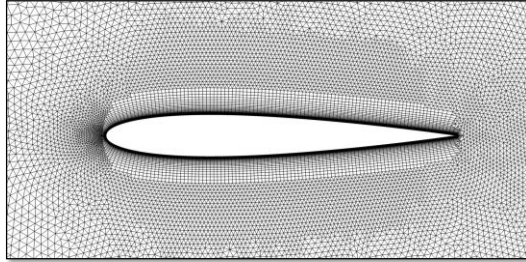
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



20

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
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



21

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_cfl_turb = 1.0 10.0
/
&global
  boundary_animation_freq = -1
/
```

Read an AFLR3 grid
Execute in 2D mode



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



22

NACA 0012 Airfoil

FUN3D 12.7-74063 Flow started 05/18/2015 at 09:06:46 with 24 processes

(Echo of fun3d.mpl)

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



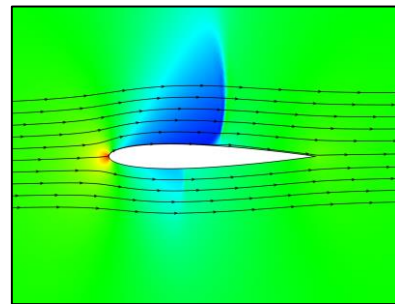
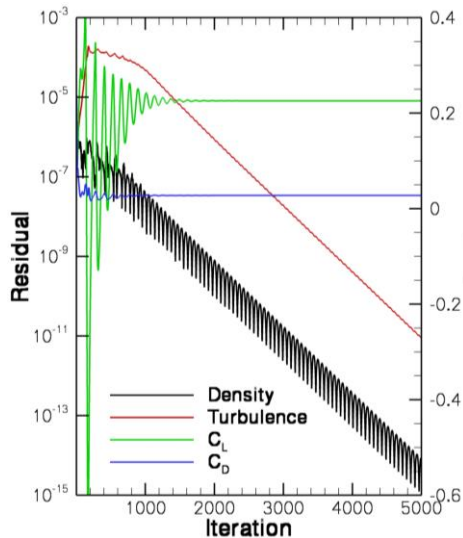
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



23

NACA 0012 Airfoil



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



24

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



25

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



26

What Could Possibly Go Wrong?

Problem

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

```
Read/Distribute Grid.
fortrl: 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?



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



27

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



28

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



29

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



30

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



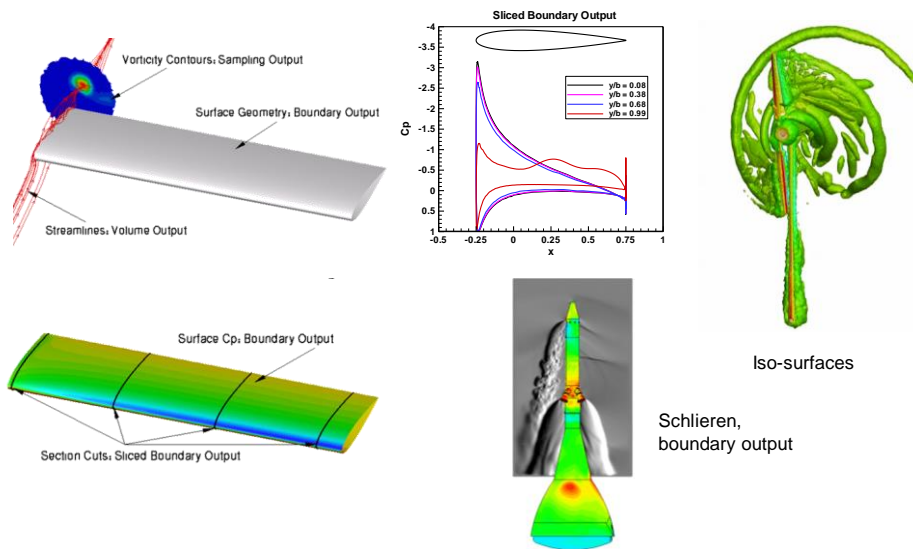
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



31

Selected Visualization Output Examples



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



32

Visualization Overview

- All of the visualization outputs require similar namelist-specified “frequency” N to activate:
 - In all cases, $N = 0, 1, 2, 3, \dots$
 - $N = 0$ 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 N^{th} time step - typically used to generate animation for unsteady flows; can also be used to observe how a steady flow converges



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



33

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 co-processing option done with a different namelist to allow “mix and match”



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



34

Visualization Overview

- For boundary surface output, default is all solid boundaries in 3D and one $y=\text{const}$ 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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



35

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



36

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 $N < 0$



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



37

Boundary Output Visualization Example

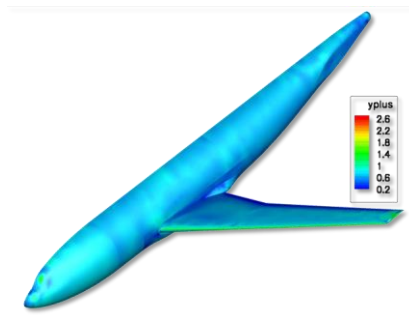
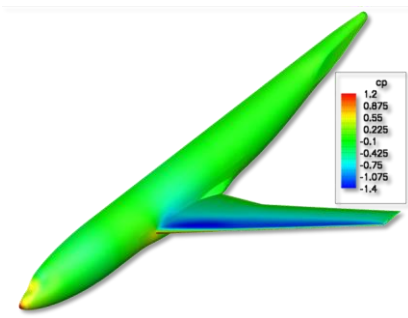
```
&global
  boundary_animation_freq = -1
/
&boundary_output_variables
  primitive_variables = .false.
  cp                  = .true.
  yplus              = .true.
/
```

Dump boundary vis at end of run

Turn off rho, u, v, w, p

Turn on C_p

Turn on y^+



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



38

Sampling Visualization Example

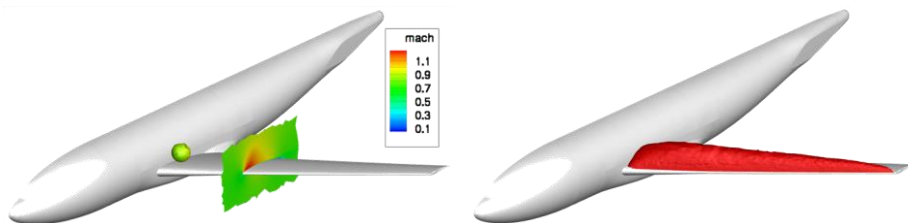
```

&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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



39

Volume Visualization Example

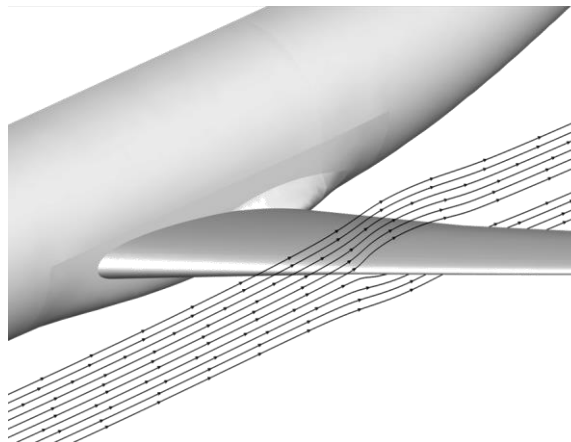
```

&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
June 20-21, 2015



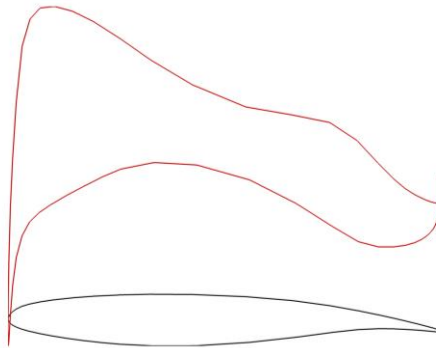
40

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
June 20-21, 2015



41

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?
 - Not currently. However, Tecplot can be easily told to load all of the files at once without having to individually select them all.
 - The team is working with Tecplot to develop their next generation of I/O API's, with special focus on massively parallel needs
 - Alternative: switch to Fieldview or CGNS output, which uses a single file



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



42

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



43

FUN3D v12.7 Training

Session 5: Boundary Conditions

Jan Carlson



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

PROBLÈMES SANS FRONTIÈRES
PROBLEMS WITHOUT BOUNDARIES

...is no problem at all...



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015




2

Boundary Conditions

...but real problems have boundaries...

- Define the problem
- Solve the problem
- Cause problems

- 
- Boundary condition list
 - Usage
 - Examples



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

Problem setup

Required files

- project name grid file: typically
`project_name[.r8,.b8].ugrid`
- namelist input: `fun3d.nml`
- boundary conditions: `project_name.mapbc`
 - Contains list of boundaries (“in order”) and the boundary condition to be associated with each one.
 - Keeping all the boundaries for a particular mesh separate (i.e. not lumping) can make for rather large and sometimes difficult to manage mapbc files.

Caveat: Not lumping boundaries, though, allows the user to retain a finer control over simulation parameters such as differing inflow/outflow conditions or transition, to name a few examples.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

Dimensionalization

Non- and otherwise

- The non-dimensionalization of the field variables, in the calorically perfect gas path, results in the ratio of Reynolds number and Mach number appearing in the transport equations.

$$\frac{\text{Re}_L}{M_\infty} = \frac{\tilde{\rho}_\infty \tilde{a}_\infty}{\tilde{\mu}_\infty} \quad \tilde{\mu}_\infty = \tilde{\mu}_{std} \left(\frac{\tilde{T}_{std} + C}{\tilde{T}_\infty + C} \right) \left(\frac{\tilde{T}_\infty}{\tilde{T}_{std}} \right)^{3/2} \quad \tilde{a}_\infty = \sqrt{\gamma R \tilde{T}_\infty}$$

- This ratio, along with the reference temperature, completely determines the flow conditions of the simulation. Tilde denotes a dimensioned parameter.

$$\tilde{u}_\infty = M_\infty \tilde{a}_\infty$$

$$\tilde{\rho}_\infty = \frac{\text{Re}_L}{\tilde{u}_\infty} \tilde{\mu}_\infty$$

$$\tilde{p}_\infty = \tilde{\rho}_\infty R \tilde{T}_\infty$$

Useful for cross-checking
auxiliary boundary
condition data.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

Boundary Condition List

- Boundary condition name (boundary condition number)
- Auxiliary data
- Limits
- Not a complete list



<http://fun3d.larc.nasa.gov>

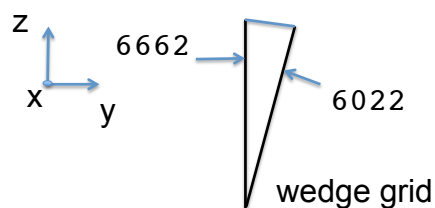
FUN3D Training Workshop
June 20-21, 2015



6

Symmetry

- `symmetry_x` (6661), y-z plane
- `symmetry_y` (6662), x-z plane
- `symmetry_z` (6663), x-y plane
- `tangency` (3000), tangential flow
- `symmetry_x_strong` (6021), zero velocity in x-mom.eqn.
- `symmetry_y_strong` (6022), zero velocity in y-mom.eqn.
- `symmetry_z_strong` (6023), zero velocity in z-mom.eqn.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

Wall

- `viscous_wall` (4000), $y^+ < 5$, $u_{wall} = v_{wall} = w_{wall} = 0$
- `viscous_weak_wall` (4110), $y^+ < 5$, τ_{wall} calculated
- `viscous_wall_function` (4100), $y^+ < 500$, τ_{wall} modeled



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

Wall

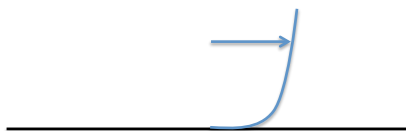
Auxiliary information

Adiabatic wall input for bc 4000

```
&boundary_conditions
  wall_temp_flag(1) = .true.
  wall_temperature(1) = -1.0
/
```

Wall function input for bc 4100

```
&turbulent_diffusion_models
  turbulence_model = 'sa','sst'
  wall_function = 'dlr'
  use_previous_utau = T
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



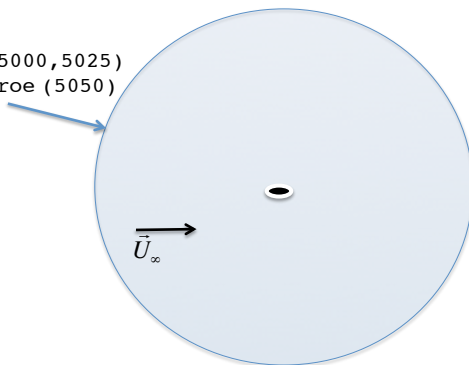
9

Farfield

Farfield boundaries use information from the `fun3d.nml` namelist parameter `mach_number (5000,5025,5050)`.

$$\rho_{\infty} = 1, u_{\infty} = Mach, v_{\infty} = 0, w_{\infty} = 0, p_{\infty} = 1/\gamma$$

```
riemann (5000,5025)
farfield_roe (5050)
```



How far is far enough?

- Problem dependent
- No gradients



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

Outflow

- Extrapolation
 - `extrapolate (5026)`, both perfect and generic gas paths
 - all 5 primitive variables extrapolated, applicable for outflow Mach ≥ 1 .
- Static pressure
 - `back_pressure (5051)`, extrapolates when local Mach ≥ 1 .
 - `subsonic_outflow_p0 (7012)`, only applicable when local Mach < 1 .
 - Auxiliary information required:

$$\text{static_pressure_ratio(ib)} = \tilde{p}_{\text{boundary}} / \tilde{p}_{\infty}$$

The static pressure ratio (SPR) is the requested static pressure on boundary `ib`, divided by the free stream static pressure.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

Inflow

- Total pressure, total temperature
 - `subsonic_inflow_pt (7011)`, both perfect and generic gas paths
 - Auxiliary information required:

$$\text{total_pressure_ratio(ib)} = \tilde{p}_{\text{total, boundary}} / \tilde{p}_{\infty}$$

$$\text{total_temperature_ratio(ib)} = \tilde{T}_{\text{total, boundary}} / \tilde{T}_{\infty}$$

- Flow direction is normal to the inflow face (default assumption).
- Applicable for inflow Mach < 1 .



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



12

Inflow

- Fixed inflow

- fixed_inflow (7100), calorically perfect gas path
- Auxiliary information required:

$$q_set(ib,1:5) = (\rho, u, v, w, p)_{boundary} \quad \text{Strictly applicable for inflow Mach} \geq 1.$$

- rcs_jet_plenum (7021), generic gas path
- Auxiliary information required: (contact Peter Gnoffo, fun3d_support)

- Massflow

- massflow_in (7036), calorically perfect gas path
- Auxiliary information required:

$$\text{massflow}(ib) = \frac{\tilde{m}_{boundary}}{\tilde{\rho}_{\infty} \tilde{a}_{\infty}}$$

$$\text{total_temperature_ratio}(ib) = \frac{\tilde{T}_{total, boundary}}{\tilde{T}_{\infty}}$$

- massflow(ib) will be in units of mesh squared



<http://fun3d.larc.nasa.gov>

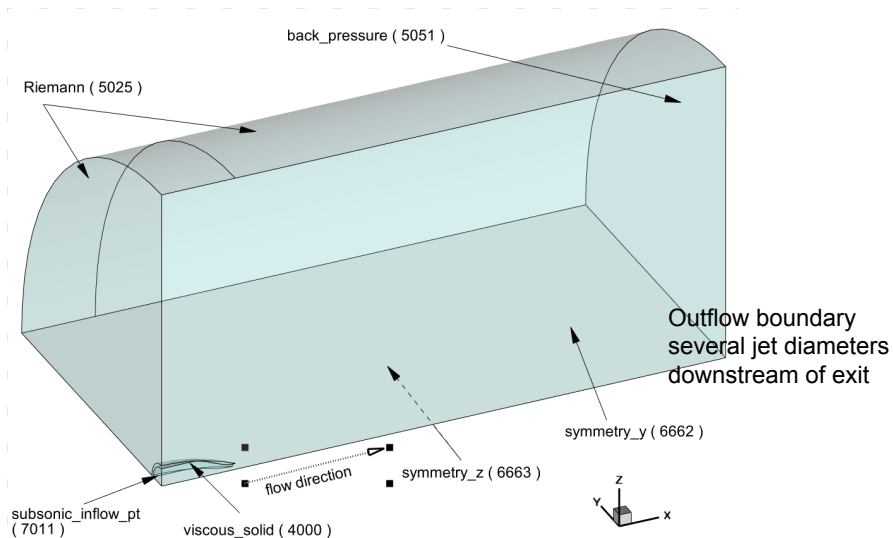
FUN3D Training Workshop
June 20-21, 2015



13

Nozzle flow strategies

Static jet case



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

Nozzle flow strategies

arn2.mapbc and fun3d.nml

```
$ cat arn2.mapbc
9 number of boundaries
1 7011 nozzle plenum inflow boundary
2 5025 farfield
3 5025 farfield
4 5051 outflow boundary
5 4000 viscous solid
6 6663 z-symmetry
7 6662 y-symmetry
8 5025 freestream inflow
9 4000 viscous solid

&boundary_conditions
total_pressure_ratio(1) = 1.357
total_temperature_ratio(1) = 1.764
static_pressure_ratio(4) = 1.0
/
```

Note: Do not lump boundaries by type, if there are several inflow or outflow boundaries that require separate settings...

Note 2: This low a pressure ratio would typically not require special volume initialization.



<http://fun3d.larc.nasa.gov>

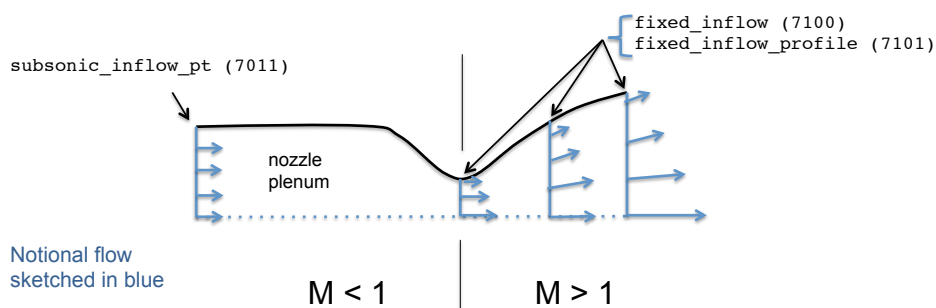
FUN3D Training Workshop
June 20-21, 2015



15

Nozzle flow strategies

Scenarios for modeling a supersonic jet



- Subsonic
- Uniform
- Typically no flow angularity
- Well posed flow state, particularly with choked flow

- Supersonic, but...
- ✗ Radial gradient due to boundary layer
- ✗ Off-axis component due to geometry



<http://fun3d.larc.nasa.gov>

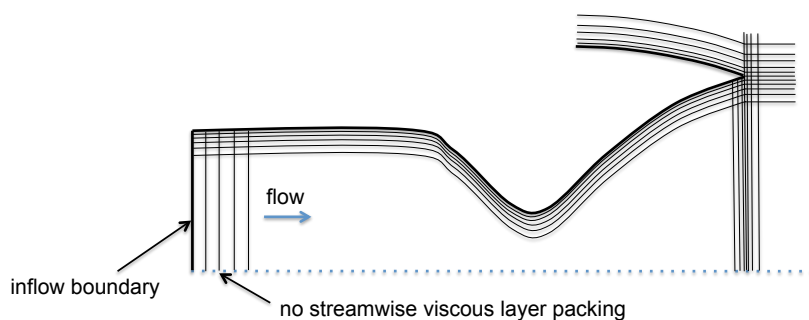
FUN3D Training Workshop
June 20-21, 2015



16

Nozzle flow strategies

Scenarios for modeling a supersonic jet



No viscous spacing in grid normal to the face
for either the subsonic or the supersonic inflow
boundaries



<http://fun3d.larc.nasa.gov>

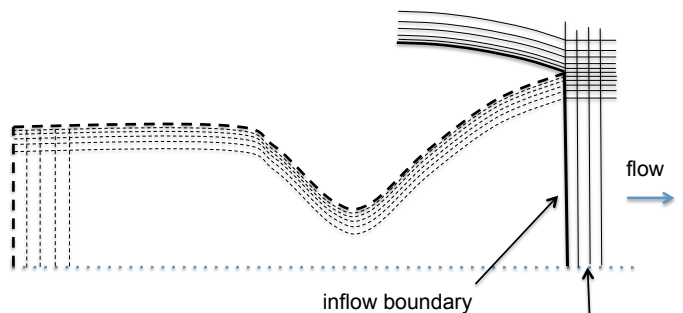
FUN3D Training Workshop
June 20-21, 2015



17

Nozzle flow strategies

Scenarios for modeling a supersonic jet



No viscous spacing in grid normal to the face
for either the subsonic or the supersonic inflow
boundaries



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

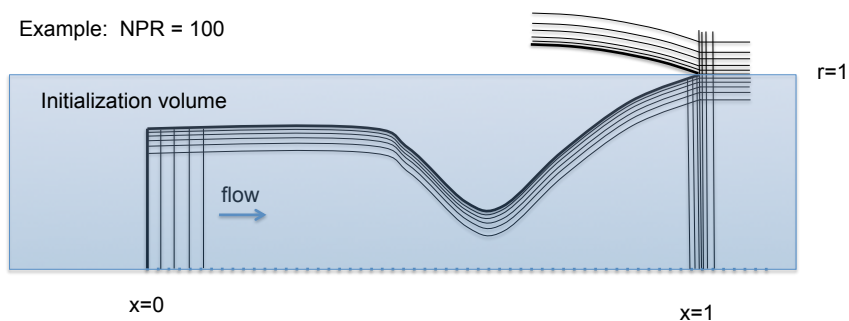


18

Nozzle flow strategies

Scenarios for modeling a supersonic jet

Example: NPR = 100



```
&flow_initialization
  number_of_volumes = 1
  type_of_volume(1) = 'cylinder'
  point1(1,:) = -0.2,0.,0.
  point2(1,:) = 1.2.,0.,0.
  radius(1) = 1.0
  rho(1) = 100.
  u(1) = 0.1
/
```

Solution startup can often be facilitated by using flow initialization



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



19

Nozzle flow strategies

fixed inflow namelist parameters

fixed_inflow (7100)

$$\rho = q_set(ib,1)$$

$$u = q_set(ib,2)$$

$$v = q_set(ib,3)$$

$$w = q_set(ib,4)$$

$$p = q_set(ib,5)$$

fixed_inflow_profile (7101)

$$\rho = \sum_{n=0}^6 profile_rho_coef(ib,n) * r^n$$

$$u = \sum_{n=0}^6 profile_u_coef(ib,n) * r^n$$

$$p = \sum_{n=0}^6 profile_p_coef(ib,n) * r^n$$

$$r = \sqrt{(p(1:3) - patch_center(ib,1:3))^2}$$



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

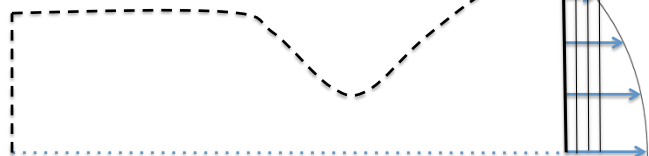


20

Nozzle flow strategies

fixed_inflow_profile

fixed_inflow_profile (7101)



Approximate
profile of a
Mach 2 jet

```
&boundary_conditions
  patch_center(1,:) = -2.0,0.,0.
  patch_scale(1) = 1.
  profile_rho_coef(1,0) = 1.
  profile_u_coef(1,0) = 2.
  profile_u_coef(1,1) = 0.
  profile_u_coef(1,2) = -1.
  profile_u_coef(1,3) = -1.
  profile_p_coef(1,0) = 0.714
/
```

This sample namelist creates a profile constant in density and pressure, and cubic in velocity, centered on (-2.,0.,0.) and physically scaled by the factor of 1. for boundary 1.



<http://fun3d.larc.nasa.gov>

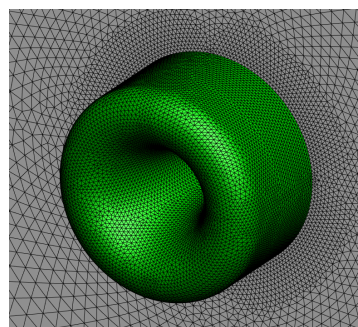
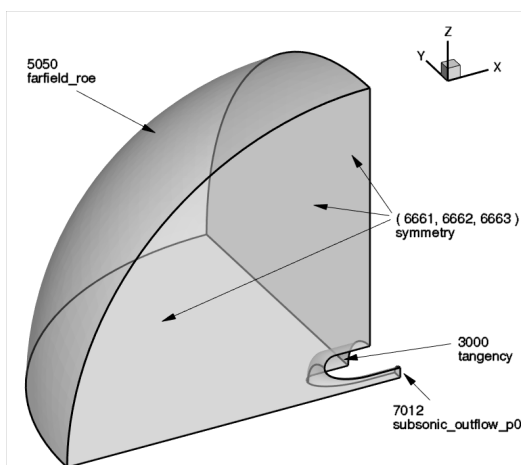
FUN3D Training Workshop
June 20-21, 2015



21

Inlet flow strategies

Bell mouth



```
&reference_physical_properties
  mach_number = 0.20
  temperature_units = 'Rankine'
  temperature = 390.0
/
&boundary_conditions
  static_pressure_ratio(1) = 0.95
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



22

Inlet flow strategies

Bell mouth

```
#Tue Apr 8 12:48:03 2008
#bell12.mapbc
Patch #      BC      Family  #surf  surfIDs      Family
-----
1 7012      1          0        0      inlet
2 5050      1          0        0      freestream inflow
3 6662      1          0        0      symmetry_y
4 6663      1          0        0      symmetry_z
5 5050      2          1        8      farfield roe
6 3000      5          0        0      bellmouth
7 3000      5          0        0      bellmouth

fun3d.nml
&governing_equations
  viscous_terms = 'inviscid'
/

&reference_physical_properties
  temperature_units = 'Rankine'
  mach_number       = 0.20
  reynolds_number    = 1.0e+5
  temperature        = 390.0
/

&boundary_conditions
  static_pressure_ratio(1) = 0.95
/
```



<http://fun3d.larc.nasa.gov>

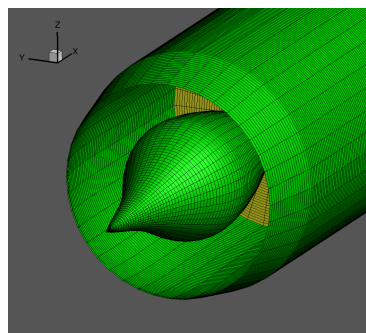
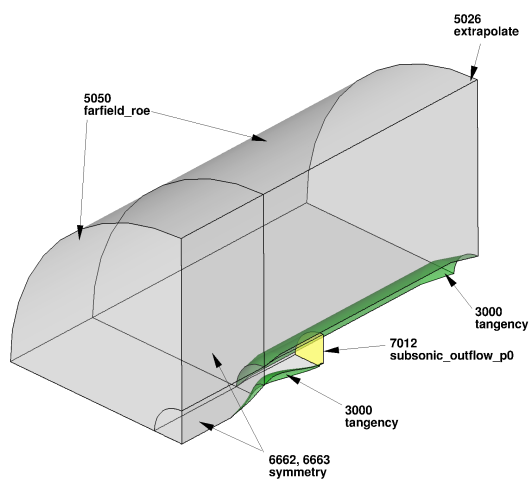
FUN3D Training Workshop
June 20-21, 2015



23

Inlet flow strategies

Supersonic inlet



```
&reference_physical_properties
  mach_number       = 1.6
  temperature_units = 'Kelvin'
  temperature        = 216.0
/

&boundary_conditions
  static_pressure_ratio(18) = 3.7
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



24

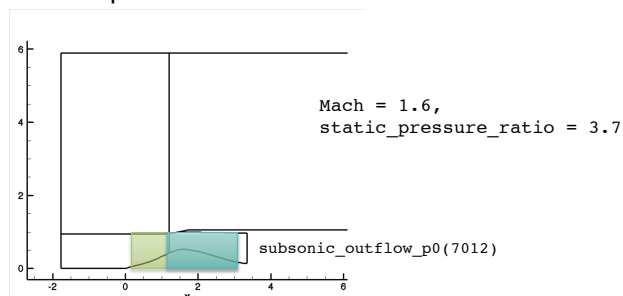
Inlet flow strategies

Supersonic inlet

Flow initialization

flow
→

```
&flow_initialization
number_of_volumes = 2
! Inlet 1
  type_of_volume(1) = 'cylinder'
  point1(:,1) = 0.0,0.,0.
  point2(:,1) = 3.0,0.,0.
  radius(1) = 0.90
  u(1) = 1.0
! Inlet 2
  type_of_volume(2) = 'cylinder'
  point1(:,2) = 1.0,0.,0.
  point2(:,2) = 3.0,0.,0.
  radius(2) = 0.90
  u(2) = 0.6
/
```



Solution startup can often be facilitated by using flow initialization. In this case, it is just about required...



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



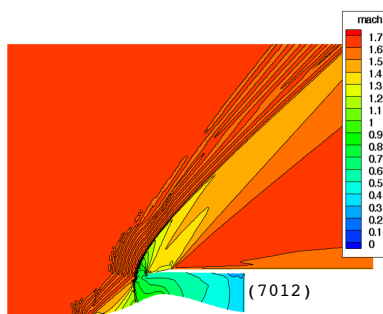
25

Inlet flow strategies

Supersonic inlet

Mach = 1.6
static_pressure_ratio = 3.7

flow
→



Additionally, aggressive CFL ramping is sometimes required. In this case, to push the shock out of the inlet.

```
&boundary_conditions
  static_pressure_ratio(18) = 3.70
/
&nonlinear_solver_parameters
  time_accuracy = 'steady'
  schedule_iteration = 1 200
  schedule_cfl = 1. 500.
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



26

Boundary conditions

- References

- [Inflow/Outflow Boundary Conditions with Application to FUN3D](#),
Jan-Renee Carlson, NASA/TM-2011-217181, October 2011.
- FUN3D V12.7 User manual

http://fun3d.larc.nasa.gov/chapter-1.html#user_manual



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



27

EOF

- Listed available boundary conditions (slightly abridged)
- Along with some typical usage
- Tips on heading off (mostly startup) problems



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



28

FUN3D v12.7 Training

Session 6: Turbulent Flow Simulations

Jan Carlson



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

Learning Goals

- Discuss some broad guidelines for turbulence models.
- List of available turbulence models (calorically perfect gas)
- Discuss the typical namelist parameters used.
- Show some sections of fun3d.nml namelists used for turbulent flow simulations.
- The detailed theory of turbulence models will not be covered in this session.
- Pros and cons of each model will not be discussed either due to time limitations.
 - All of the models will likely work some of the time.
 - But none of the models will work all of the time.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

The List

Steady flow simulations

- One-equation
 - Spalart-Allmaras (*sa*), Recherche Aerospatiale, No. 1, 1994.
 - Negative Spalart-Allmaras (*sa-neg*), ICCFD7-1902, 2012.
- Two-equation
 - Menter-SST (*sst*), AIAAJ (32), 1994.
 - Menter-SST with vorticity source term (*sst-v*), NASA-TM-103975, 1992.
 - Menter-SST from 2003 (*sst-2003*), Turbulence, Heat and Mass Transfer 4.
 - Wilcox k-omega (*wilcox2006*), AIAAJ (46), 2008.
 - Wilcox k-omega (*wilcox1998*), Turbulence Modeling for CFD, 1998.
 - Wilcox k-omega (*wilcox1988*), AIAAJ (26), 1988.
 - Nonlinear k-omega (*EASMKo2003-s*), J Aircraft (38), 2001.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

The List

Steady flow simulations

- Four-equation
 - Langtry-Menter transition model (*gamma-ret-sst*), AIAA-2005-0522.
- Seven-equation
 - Wilcox Stress-omega RSM (*wilcoxRSM-w2006*), Turbulence Modeling for CFD, 2006.
 - SSGLRR-RSM (*SSGLRR-RSM-w2012*), AIAA Journal, Vol. 53, No. 3, 2015, pp. 739-755.

Other references and detailed explanations of the models can be found at the turbulence modeling website:

<http://turbmodels.larc.nasa.gov>



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

The List

Time accurate flow simulations

- One-equation
 - Detached eddy simulations, (des, des-neg), TCFD (20), 2006.
- Two-equation
 - Hybrid RANS-LES (hrles), AIAA-2008-3854.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

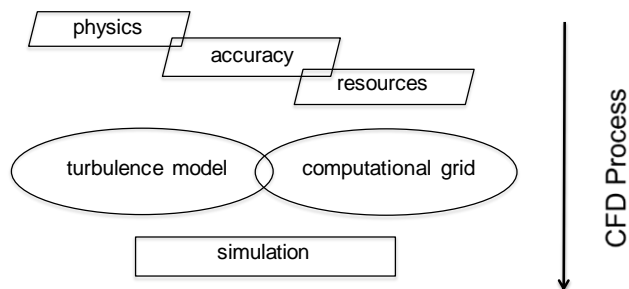


5

General usage guidelines

Do we even need to perform a turbulent flow simulation?

- Flow physics
 - What physics need to be simulated/predicted?
 - high speed flow -> *possibly* largely laminar
 - corner flow -> *possibly* anisotropic turbulence
 - blunt body wake -> *possibly* large eddy simulations
- Computational requirements
 - to evaluate the grid's resolution required for a certain accuracy



<http://www.stanford.edu/class/me469b/handouts/turbulence.pdf>, slide 51



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

General usage guidelines

- Appropriate spacing of the mesh on viscous solid walls must be used.
 - Generally accepted spacing is between .1 and 2.5 wall units.
 - Using wall functions, generally accepted spacing is between 0.1 and 250 wall units.
 - Many problems may have multiple scales, so no one physical distance for the first node spacing will suit the whole problem.
- Generate a mesh with appropriate resolution to model the problem (within the limits of the available computational resources).
 - Try not to expand the mesh spacing too quickly away from a viscous wall.
 - Typically the more curvature in the physical geometry, the higher concentration of mesh.
- One-equation models like Spalart-Allmaras tend to be very robust, cover a very wide range of flow situations and are a compromise between simplicity and accuracy.
- Multi-equation models like the Menter-SST or RSM require more computational resources, but are more physically complete and can, possibly, add more accuracy to the solution...though YMMV.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

General usage guidelines

- Solutions to a steady state are adequate for many problems.
- Depending upon the physics of the simulation, though, time-accurate solutions may be required.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

Namelists

fun3d.nml

For turbulent flow simulations, depending upon the turbulence model and problem the following namelists within fun3d.nml are used.

- `&governing_equations`
- `&turbulent_diffusion_models`
- `&spalart`
- `&gammaretsst`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



9

Spalart-Allmaras

fun3d.nml

```
&governing_equations
  eqn_type      = 'cal_per_compress'
  viscous_terms = 'turbulent'
/

&turbulent_diffusion_models
  turbulence_model = 'sa' !default
  ! current 1-eqn options: 'sa-neg', 'des', 'des-neg'
  turb_compress_model = 'none'
  ! current options: 'ssz' ! (Ref. AIAA-95-0863, Shur et al.)
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

Spalart-Allmaras

fun3d.nml

```
&spalart
  turbinf      = 3.0
    ! free stream value for spalart model
  ddes         = .false.
    ! for activating delayed DES model
  ddes_mod1    = .false.
    ! Mod to DDES, Ref. AIAA Paper 2010-4001
  sarc         = .false.
    ! Ref. AIAAJ, Vol.38, No.5, 2000, pp.784-792.
  sarc_cr3     = 1.0
    ! constant associated with SARC model
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

Menter-SST

fun3d.nml

```
&governing_equations
  eqn_type      = 'cal_per_compress'
  viscous_terms = 'turbulent'
/
&turbulent_diffusion_models
  turbulence_model = 'sst'
!other options: 'sst-v', 'sst-2003', 'gamma-ret-sst'
! 'hrles'
/
&gammaretsst
  set_k_inf_w_turb_intsty_percent = 0.2 ! (percent)
  set_w_inf_w_eddyviscosity       = 1.0 ! (nondim)
  transition_4eqn_on               = .true.
    ! toggles transition
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



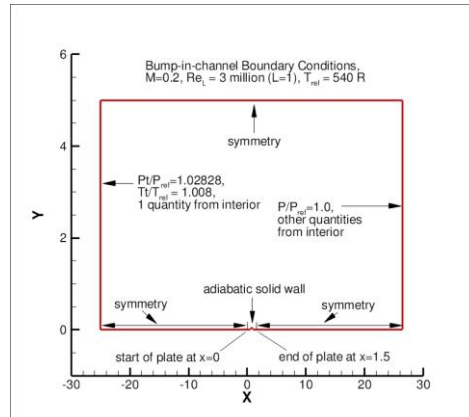
12

Sample fun3d.nml

Subsonic bump using S-A

<http://turbmodels.larc.nasa.gov/bump.html>

```
&project
  project_rootname = 'bump_3levelsdown_177x81'
/
&reference_physical_properties
  mach_number      = 0.2
  reynolds_number   = 3000000.0
  temperature       = 540.0
  temperature_units = 'Rankine'
/
&turbulent_diffusion_models
  turbulence_model = 'sa'
/
&nonlinear_solver_parameters
  schedule_iteration = 1 250
  schedule_cfl       = 10. 250.
  schedule_cfl_turb  = 10. 250.
/
&boundary_conditions
  total_pressure_ratio(3) = 1.02828
  total_temperature_ratio(3) = 1.008
  static_pressure_ratio(4) = 1.0
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

Sample fun3d.nml

Time accurate simulation using a S-A based DES model

```
&turbulent_diffusion_models
  turbulence_model = 'des'
/
&nonlinear_solver_parameters
  time_accuracy      = '2ndorderOPT'
  time_step_nondim   = 0.10
  pseudo_time_stepping = 'on'
  subiterations      = 10
  schedule_iteration  = 1 100
  schedule_cfl        = 5. 5.
  schedule_cfl_turb   = 5. 5
/
```

Details of running a time accurate simulations are covered in Session 11.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

EOF

Turbulent flow simulations with Fun3D

Several turbulence model options are available in V12.7

Namelist nomenclature has been discussed.

Caveats:

Meshing and turbulence model decisions are highly dependent on the degree of fidelity and accuracy desired.

The desired aspects, though, may not fit inside the resources available.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



15

FUN3D v12.7 Training

Session 7: Supersonic and Hypersonic Perfect Gas Simulation

Mike Park



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

Session Overview

- How to use FUN3D to compute **perfect gas** supersonic and hypersonic flows (`eqn_type="compressible"`)
 - What are the challenges and strategies
 - Inviscid flux types and inviscid flux gradient limiters options that work the best for supersonic and hypersonic flows
 - Required practice for running adjoint with gradient limiters for design and grid adaptation
- Methods to initialize supersonic and hypersonic flows
- Example of a hypersonic flow application
- What to do when things go wrong
- The focus is on high-speed flows, but the strategies discussed can be used in other flow regimes



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

Perfect and Generic Gas Simulation

- The input parameters described in this talk are only valid for (eqn_type="compressible")
- Generic gas input parameters are different, but the philosophy is similar
- Work is underway to merge the options where possible, but consult generic gas specific documentation for details



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

What Are the Challenges?

- The inviscid terms can be discontinuous, i.e., when there are shocks
 - Entropy problem: strong shocks can cause difficulties in inviscid flux schemes especially near points in the flow where the dissipation vanishes
 - Monotonicity problem: shocks cause discontinuities that make robust implementation of higher order schemes difficult
- The inviscid terms can be a problem when there is strong expansion
 - Positivity problem: strong expansions can cause difficulties such that the local conditions approach a vacuum
 - Sonic rarfaction or “expansion shock” problem: strong expansions near the sonic point where dissipation due to the u-a eigenvalues vanishes can cause difficulties
- Turbulence modeling challenges compound these issues but are not the focus of this talk



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

Inviscid Flux Types

- Inviscid flux schemes fall into several categories:
 - Contact preserving, i.e., good for viscous flows
 - Flux difference splitting scheme of `flux_construction = "roe"`
 - Non positivity near vacuum conditions
 - The sonic rarefaction problem
 - The "carbuncle" problem
 - Non preservation of the total enthalpy in shocks
 - Entropy fixes (Eigenvalue smoothing) exist for some but not all of these problems
 - Flux splitting schemes such as `flux_construction = "hllc"` and `"ldfss"` may display some limited unphysical behavior at very strong normal shocks
 - Non-contact preserving, i.e. not usually good for viscous flows
 - Flux vector split scheme, `flux_construction = "vanleer"`, has desirable qualities
 - Positivity near vacuum conditions
 - Preservation of the total enthalpy in shocks



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

Inviscid Flux Types

- Inviscid flux schemes fall into several categories:
 - Hybrid or "blended" schemes
 - The `flux_construction = "dldfss"` scheme is a blend of two schemes
 - The `vanleer` scheme at shocks via a shock detector
 - The `ldfss` scheme near walls via a shock and boundary layer detector



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

Inviscid Flux Gradient Limiter Types

- Gradient limiters are available in two types:
 - Edge based : limiting is done on an edge by edge basis,
`flux_limiter = "minmod", "vanleer", "vanalbada" and "smooth"`
 - They are less dissipative and they work pretty well on hex grids but they are not as robust on mixed element or tetrahedral grids.
 - They are not "freezable" and may cause convergence to get hung up by limiter cycling. They also can not be used when using the adjoint solvers
 - Stencil based : limiting is done based on the max and min reconstructed higher order edge gradients that exist over the entire control volume
`"stencil", flux_limiter = "barth", "hvanleer", "hvanalbada", "hsmooth", and "venkat"`
 - They are more robust but more dissipative and work on all grid types
 - They are "freezable", i.e. they can be frozen after a suitable number of iterations which sometimes will allow the solution to converge further
 - They must be frozen when solving adjoint equations
 - Limiters with the "h" prefix include a heuristic stencil based pressure limiter to increase robustness



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

Realizability

- Nonphysical (negative density or pressure) reconstructions are set to cell averages (first order) accompanied with a "realizability" warning
- Nonlinear density and pressure updates are floored to a ratio of freestream with the `f_allow_minimum_m` namelist variable
 - The default floor may need to be lowered if the simulation requires it



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

Calorically Perfect Supersonic Flow

- Maximum Mach number in computational domain < 3.0 such that:
 - Shocks are relatively weak
 - Expansion fans are relatively weak
- Inviscid flux options suitable for these applications:
 - When Euler: `viscous_terms = "inviscid"`
 - `flux_construction = "vanleer", "ldfss", "hlhc" or "roe"`
 - When Navier-Stokes: `viscous_terms = "laminar" or "turbulent"`
 - `flux_construction = "ldfss", "hlhc", or "roe"`
- Inviscid flux gradient limiter options most suitable for these applications:
 - `flux_limiter = "vanleer", "vanalbada", "hvanleer", or "hvanalbada"`
- For applications that require solving the adjoint:
 - `flux_construction = "vanleer" or "roe"`
 - `flux_limiter = "hvanleer" or "hvanalbada"`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



9

Calorically Perfect Hypersonic Flow

- Maximum Mach number in computational domain > 3.0 such that:
 - Shocks may be strong, especially when there are normal shocks
 - Expansion fans may be strong
- Inviscid flux options suitable for these applications:
 - When Euler: `viscous_terms = "inviscid"`
 - `flux_construction = "vanleer" or "dldfss"`
 - When Navier-Stokes: `viscous_terms = "laminar" or "turbulent"`
 - `flux_construction = "dldfss"`
- Inviscid flux gradient limiter options most suitable for these applications:
 - `flux_limiter = "hvanleer" or "hvanalbada"`
- For applications that require solving the adjoint:
 - `flux_construction = "vanleer" or "roe"`
 - `flux_limiter = "hvanleer" or "hvanalbada"`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

Nonlinear Equations

- When solving nonlinear equations (e.g., Euler, Navier-Stokes), the initial guess is critical!
- Transients can be much more challenging than the steady solution
 - Solution under and over shoots can be aggravated
 - Nonphysical states may be transited
 - Boundary conditions are less robust with large gradients nearby
 - Linear system solution scheme and nonlinear defect correction solution schemes can become unstable



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

Strategy

- Perform the simulation in phases
 - Initialization
 - Target solution scheme
 - Optional end game that freezes limiter for better iterative convergence.
- Initialization is the primary challenge to success for high speed, internal, and propulsion flows



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



12

Initialization Strategies

- The default initialization fills the domain with freestream flow and applies strong boundary conditions
 - Creates high gradients adjacent to the boundary
 - Sets up an unphysical expansion on backward facing surfaces
- The goal of initialization is to improve this default flow field with one that establishes the physical mechanisms of the simulations (e.g., boundary layers, shear layers, recirculation zones)
 - Moves large gradient regions away from the boundaries and into the interior of the domain
- You have the freedom to use methods that are inaccurate as long as you later restart the solution with an appropriate method for your simulation
 - Includes changing boundary conditions, freestream conditions, etc.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

Initialization Strategies

- Use `first_order_iterations` to create a spatially first-order solution
 - This helps the nonlinear update because there are less approximations in defect correction
- Use a more dissipative flux scheme
 - Roe with excessive Eigenvalue smoothing
 - `rhs_u_eigenvalue_coef`, `lhs_u_eigenvalue_coef`,
`rhs_a_eigenvalue_coef`, `lhs_a_eigenvalue_coef`
 - "vanleer" for Navier-Stokes
- Restart from a lower Mach number or angle of attack solution
- Slow down (lower CFL number or physical time step)
 - This aids the stability of the linear solve and nonlinear updates
- Combinations of these strategies



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

Initialization Strategies

- Explicitly initialize with the `&flow_initialization` namelist
 - Fill plenums with subsonic high density and pressure gas
 - Place a subsonic wake behind an aft facing step
 - Surround the entire vehicle with a sphere of post shock flow conditions (subsonic high density and pressure gas)
 - May reduce the execution time by allowing the use of larger CFL numbers



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



15

Solution Scheme

- See the advantages and disadvantages of the available fluxes and limiters
- Adjust (ramp) the CFL number for the best convergence rate
- Expect the solution convergence to stall due to limiter buzz



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



16

End Game

- Optionally freeze the gradient limiter to overcome limiter buzz
 - Make sure the solution is sufficiently converged



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



17

Multiple Step Approach

- Applications with shocks and expansions may need to be run in multiple steps
 - Step 1 : Run solution first order while scheduling the CFL number to evolve the solution to a quasi-steady state;
 - Initialize the flow appropriately
 - Set `first_order_iterations` to the same as the number of iterations specified by steps
 - Use `schedule_iteration`, `schedule_cfl`, and `schedule_cfl_turb` to slowly increase CFL number
 - Step 2 : Restart solution higher order while scheduling the CFL number to compute the final solution;
 - Read the restart file, i.e. `restart_read = "on"`
 - Set `first_order_iterations = 0`
 - The CFL ramping of `schedule_iteration`, `schedule_cfl`, and `schedule_cfl_turb` may need to be less aggressive



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

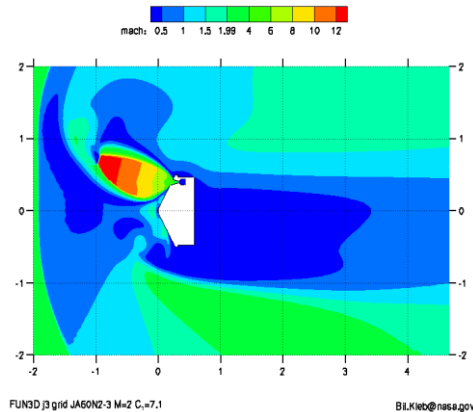


18

Supersonic/Hypersonic Retro-propulsion Flow Example

- Turbulent retro-propulsion re-entry plume flow in one run that includes the three phases
- Relevant namelist settings

```
&code_run_control
  steps      = 7500
  restart_read = 'off'
/
&inviscid_flux_method
  first_order_iterations = 2500
  freeze_limiter_interation = 5000
  flux_limiter = 'hvanalbada'
  flux_construction = 'dldfss'
/
&nonlinear_solver_parameters
  schedule_iteration = 1 100
  schedule_cfl       = 0.1 10.
  schedule_cfl_turb  = 0.01 1.
/
```



<http://fun3d.larc.nasa.gov>

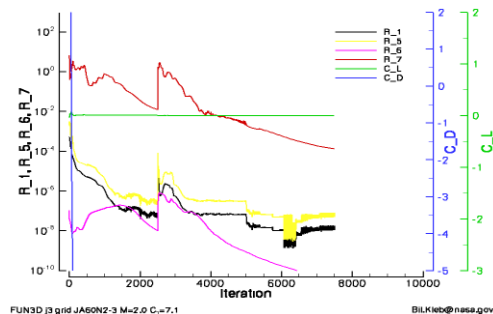
FUN3D Training Workshop
June 20-21, 2015



19

Supersonic/Hypersonic Retro-propulsion Flow Example

- Switch from 1st order to 2nd order scheme occurs at 2500 iterations
- The hvanalbada limiter was frozen at 5000 iterations
- Continuity and energy equation residuals converged ~ 4 orders
 - Jet unsteadiness probably preventing further convergence
- Lift has converged, i.e. is no longer changing



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



20

Supersonic/Hypersonic Retro-propulsion Flow Example Some Observations

- Turbulent flow has made this case easier to run because of the added dissipation caused by the eddy viscosity in the retro-propulsion jet
- If this case were laminar, it would probably be more difficult to run
 - You would need to be careful that the `dldfss` flux scheme does not add too much dissipation by refining the grid
 - You may need to resort to a multiple step running approach or explicit initialization of the flow field



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



21

Diagnosis When Things Go Wrong

- Restart the solution and visualize just before an increase in the residual
- Create movies near the largest residual location
- Try to isolate the problem location
- Check your grid resolution near the maximum residual location
 - Under-resolved expansions can cause a lot of trouble
 - Really large grid aspect ratios near expansions can cause trouble
- Check to make sure your boundary conditions are well posed
 - This is especially true for internal flows



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



22

Diagnosis When Things Go Wrong

- Isolate the problem to linear system or nonlinear update
 - Invoke the `--monitor_linear` command line option
 - Set `linear_projection = .true.` or change the number of linear sweeps
 - Lowering CFL number can aid linear and nonlinear stability
 - Try a different initialization strategy



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



23

What We Learned

- Recommended use cases and descriptions of flux schemes
- Recommended use cases for gradient limiters and how to freeze them
- Initialization strategies
- What the convergence behavior may look like
- What to do when things go wrong



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



24

FUN3D v12.7 Training

Session 8:

Parameterization Tools

Bill Jones



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

Setting

- FUN3D shape design relies on a pre-defined relationship between a set of parameters, or design variables, and the discrete surface mesh coordinates
- Given DV , surface parameterization determines X_{surf}
 - 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 mesh points) to dozens or hundreds
 - Optimizers will perform more efficiently
 - Smoother design space
- An additional requirement of the parameterization package is that it provides the Jacobian of the relationship between the design variables and the surface mesh, $\partial X_{surf} / \partial DV$
- 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 FFD based tool
 - Sculptor®: Commercial package from Optimal Solutions



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

Learning Goals

- Parameterize geometry with respect to DVs to control shape
 - MASSOUD
 - BandAids
- Generate perturbed surface mesh and SDs for FUN3D design
 - Visual validation
- What we will not cover
 - Body transformations
 - How to use the data in FUN3D
 - That will be covered in the next session



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

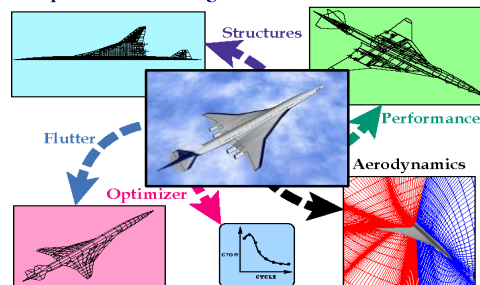


3

MASSOUD

- **Multidisciplinary Aerodynamic-Structural Shape Optimization Using Deformation**
 - AIAA-2000-4911 (Jamshid Samareh)
- Used to generate consistent models for MDAO
 - Same shape changes communicated across all disciplines
- Highly tailored for aerodynamic shapes
 - Parameters familiar to engineer
- Mesh based parameterization

Multidisciplinary Aero/Structural Shape Optimization Using Deformation (MASSOUD)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

MASSOUD Key Ideas

- Uses soft object animation algorithms for deforming meshes
 - Nonlinear global deformation (twist and dihedral)
 - NURBS surface (camber and thickness)
 - Free-form deformation (planform)
- Parameterizes the discipline meshes
 - Avoids mesh regeneration
- Parameterizes the changes in shape, not the shape itself
 - No need to reproduce shape
 - Reduces the number of design variables



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

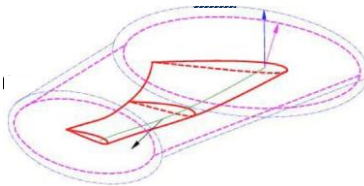


5

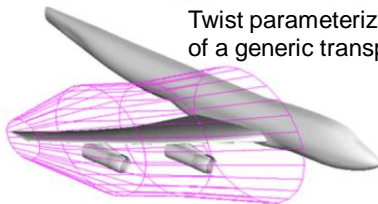
MASSOUD Twist and Shear

- Nonlinear Global Deformation
 - Wrapped in twist cylinder
 - Twisted and sheared in planes along span normal to twist vector

Twist parameterization
of a generic wing



Twist parameterization
of a generic transport



Extreme deformation
of a generic transport



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

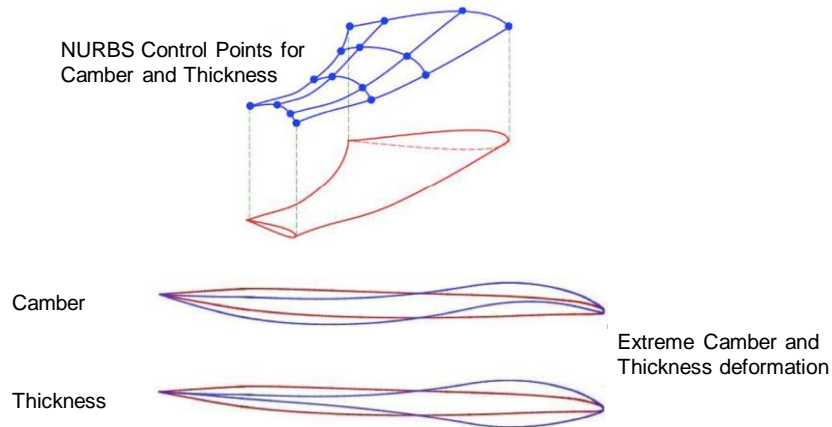


6

MASSOUD Camber and Thickness

- Non-Uniform Rational B-Spline (NURBS)
 - Represents the shape changes not the shape

NURBS Control Points for
Camber and Thickness



<http://fun3d.larc.nasa.gov>

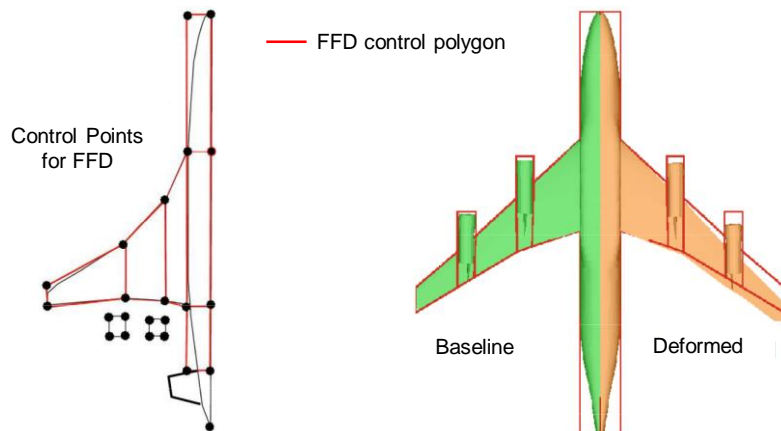
FUN3D Training Workshop
June 20-21, 2015



7

MASSOUD Planform

- Free-form Deformation (FFD)
 - Surround shapes with quadrilaterals



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

MASSOUD Installation

- Distributed as source code
 - Single **Makefile** uses GNU C compiler (**gcc**)
 - Any localization must be done manually
- Creates two executables
 - **massoudDesignDriver** creates parameterization
 - **massoud** surface mesh perturbation with sensitivity data



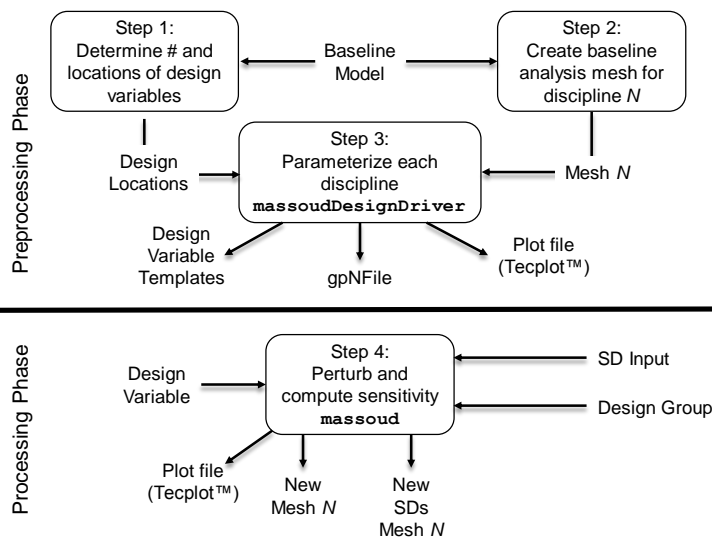
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



9

MASSOUD Process



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

MASSOUD Step 1

- Parameterization requires input to define DV locations
 - Small ASCII file
 - Contains 7 groups of oriented curves
 - X axis is positive downstream
 - Y is positive out the wing span
 - Y should be positive with curves monotonically increasing
 - GridTool can be used to create the file



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

MASSOUD Design Locations File

```

Design location file Case Name Title (SECTION 1)
np ne ntwist ncmax
4 1 2 100 0 1 2

Pt X Y Z (SECTION 2)
0 -0.0010000 -1.0010000e+00 0.0000000e+00
1 1.0010000 -1.0010000e+00 0.0000000e+00
2 1.0010000 0.0000000e+00 0.0000000e+00
3 -0.0010000 0.0000000e+00 0.0000000e+00
0 1 2 3

#Twist Vector (SECTION 3)
# Ax Ay Az
0.0000000e+00 1.0000000e+00 0.0000000e+00
# x y z ir or
2.5000000e-01 -1.0000000e+00 0.0000000e+00 1000.0 10000.0
2.5000000e-01 0.0000000e+00 0.0000000e+00 1000.0 10000.0

#Le/Te definitions (SECTION 4)
2
0.0000000e+00 -1.0010000e+00 0.0000000e+00
0.0000000e+00 0.0000000e+00 0.0000000e+00
2
1.0000000e+00 -1.0010000e+00 0.0000000e+00
1.0000000e+00 0.0000000e+00 0.0000000e+00
5 2 0.0000000e+00 -1.0010000e+00 0.0000000e+00 1.0000000e+00 # Thickness (SECTION 5)
0.0 0.0000000e+00 0.0000000e+00
0.1 0.0000000e+00 0.0000000e+00
0.5 0.0000000e+00 0.0000000e+00
0.75 0.0000000e+00 0.0000000e+00
1.0 0.0000000e+00 0.0000000e+00
3 2
0.0000000e+00 -1.0010000e+00 0.0000000e+00
0.0000000e+00 -0.5000000e+00 0.0000000e+00
0.0000000e+00 0.0000000e+00 0.0000000e+00
5 2 0.0000000e+00 -1.0010000e+00 0.0000000e+00 1.0000000e+00 # Camber (SECTION 6)
0.0 0.0000000e+00 0.0000000e+00
0.1 0.0000000e+00 0.0000000e+00
0.5 0.0000000e+00 0.0000000e+00
0.75 0.0000000e+00 0.0000000e+00
1.0 0.0000000e+00 0.0000000e+00
3 2
0.0000000e+00 -1.0010000e+00 0.0000000e+00
0.0000000e+00 -0.5000000e+00 0.0000000e+00
0.0000000e+00 0.0000000e+00 0.0000000e+00

```

Planform

Twist

Leading and
Trailing Edges

Thickness

Camber



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

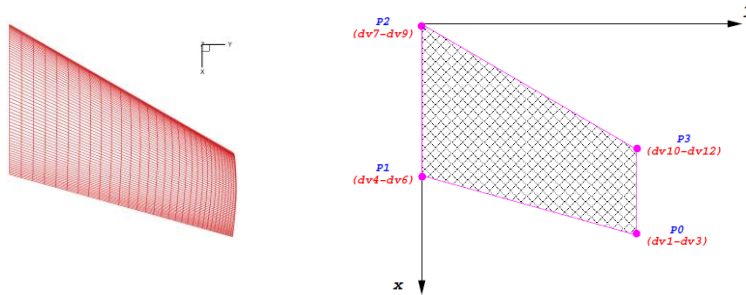


12

MASSOUD Design Locations

1. Planform

- Cover planform with 5 point quadrilaterals
 - Closed but orientation does not matter
- 1 Curve per planform section
- GridTool Family name “**planform**”



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

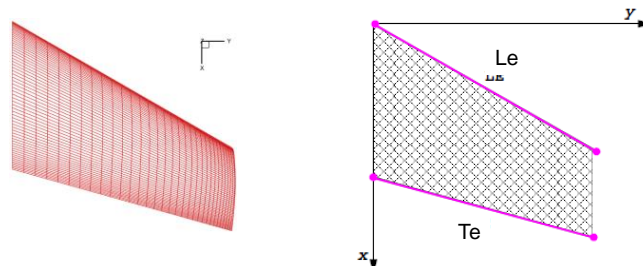
MASSOUD Design Locations

2. Leading Edge

- Create an n point PWL curve defining the leading edge
 - Must bound all mesh nodes
 - May extend beyond actual geometry
- GridTool Family name “**le**”

3. Trailing Edge

- Create an n point PWL curve defining the trailing edge
 - Must bound all mesh nodes
 - GridTool Family name “**te**”



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

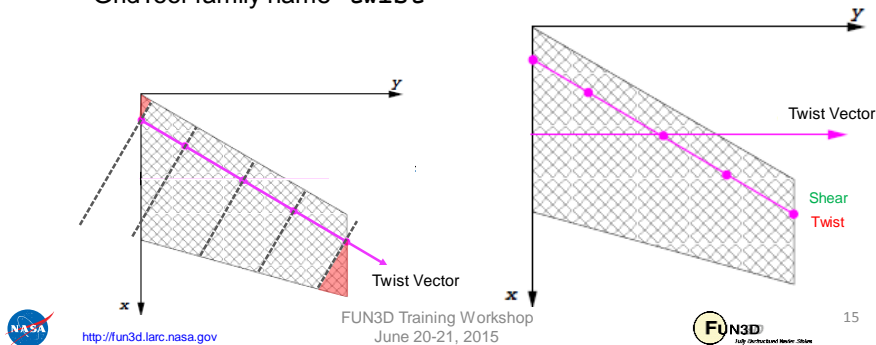
MASSOUD Design Locations

4. Twist Vector

- Create a 2 point curve to represent the twist vector
 - Twist sections defined normal to this vector
- GridTool Family name “**twistv**”

5. Twist Location

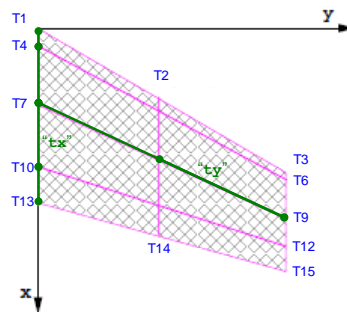
- Create an n point PWL curve to represent the n twist locations
- Airfoil sections defined at these points normal to “**twistv**”
 - First and last section must bound the Y coordinates of the target mesh
- GridTool family name “**twist**”



MASSOUD Design Locations

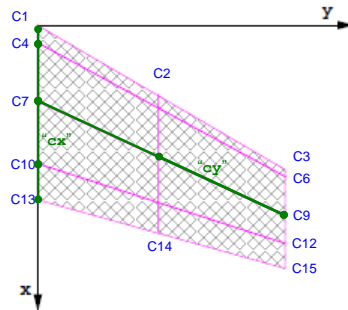
• Thickness

- Chordwise
 - Create an n point PWL curve to represent the n chordwise thickness locations
 - Start, length, and %
 - GridTool family name “**tx**”
- Spanwise
 - Create an m point PWL curve to represent the m spanwise thickness locations
 - Should bound Y values of all target mesh nodes
 - Beginning and ending Y coordinates must be bounded by the Y coordinates of **both** the “**le**” and “**te**” curves
 - May be a duplicate of the “**twist**” curve
 - GridTool family name “**ty**”
- $n \times m$ set of DVs



MASSOUD Design Locations

- Camber
 - Same as for Thickness but with GridTool family names “**cx**” and “**cy**” respectively
 - May be duplicates of “**tx**” and “**ty**”
 - Two curves define $n \times m$ set of DVs



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



17

MASSOUD Step 2

- Dump out surface meshes of interest in a Tecplot™ format
 - Includes the surface node coordinates
 - Global ID of the surface nodes wrt the volume mesh
 - FUN3D flow solver CLO ‘--write_massoud_file’
 - Produces “[project]_massoud_bndryN.dat” file for body N
 - Default extracts all viscous boundary surfaces as separate bodies

- FUN3D Namelist controls

```
&massoud_output
  n_bodies      = 2      ! Parameterize 2 bodies
  nbndry(1)     = 6      ! 1st body has 6 boundaries
  boundary_list(1) = '3-8' ! Boundaries in 1st body
  nbndry(2)     = 3      ! 2nd body has 3 boundaries
  boundary_list(2) = '9,10,12' ! Boundaries in 2nd body
/
```

- **boundary_list()** indices should reflect boundary lumping



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



18

MASSOUD Step 3

- Generate geometry parameterization

```
% massoudDesignDriver -t input_massoud_bndry1.dat \
                        designLocations \
                        design_gp.1
```

- Geometry parameterization is output in “**design_gp.1**”
 - Used as input to `massoud`
- Additional output
 - “**designVariableTemplate**”
 - Reference for “**design.1**” file with zero perturbations
 - “**designTemplate.usd**”
 - Reference for “**design.usd.1**” user defined variable links
 - “**designVariableTemplateNumber**”
 - Lists the DV indices by DV type (planform, twist, etc.)
 - “**baselineShape.plt**”
 - Tecplot™ readable zero perturbation reference
 - Errors in “**GP.log**”



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



19

MASSOUD Step 4

- Mesh deformation %**massoud massoud.N**
 - Where MASSOUD input is in “**massoud.N**”
 - FUN3D design will utilize “**customDV.N**” for perturbations

```
#MASSOUD INPUT FILE
# Option (0 analysis), (> 0 sd using user dvs) (-1, sd using massoud dvs)
-1
# 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 Option > 0)
design.usd.1
# output file mesh 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]
```



<http://fun3d.larc.nasa.gov>

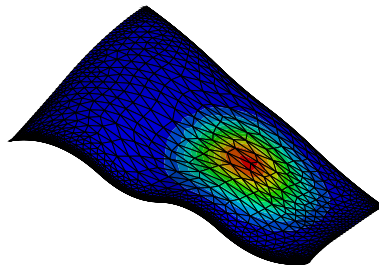
FUN3D Training Workshop
June 20-21, 2015



20

MASSOUD Results

- Visual inspection
 - Tecplot™
 - “model.tec.1.sd1” contains mesh and SDs
 - (e.g. XD1, YD1, ZD1... XDndv, YDndv, ZDndv)
 - GridTool
 - % `GridTool -d model.tec.1.sd1`
 - Sliders to interactively perturb DVs
 - Twist is non-linear and is only indication of change



<http://fun3d.larc.nasa.gov>

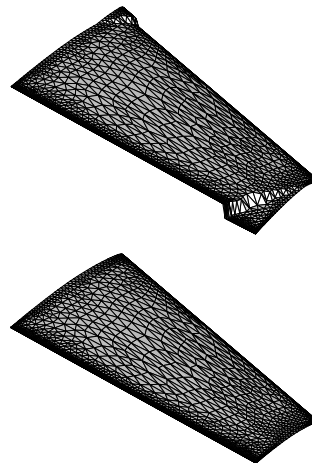
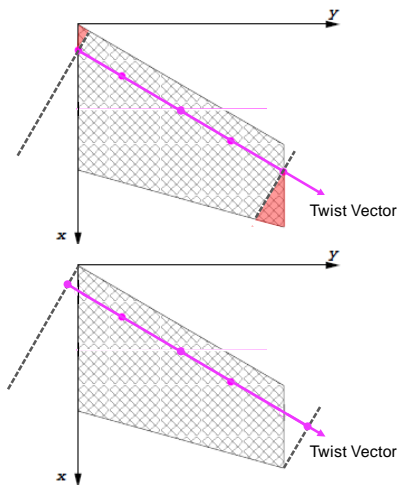
FUN3D Training Workshop
June 20-21, 2015



21

What Could Go Wrong (1 of 2)

- Failure ... check “GP.log”
- Design locations must be defined to bound all target mesh nodes



<http://fun3d.larc.nasa.gov>

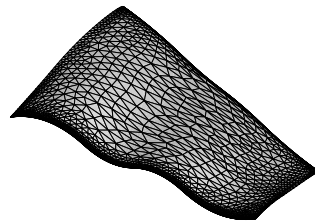
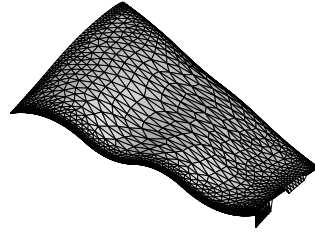
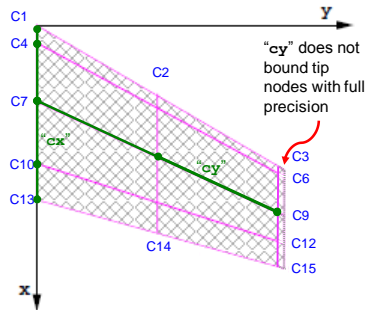
FUN3D Training Workshop
June 20-21, 2015



22

What Could Go Wrong (2 of 2)

- Design locations must be defined to bound all target mesh nodes



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



23

MASSOUD User Defined Variables

- New variables as linear combination of MASSOUD variables

$$\frac{\partial \bar{R}}{\partial V_j} = \frac{\partial \bar{R}}{\partial P_i} \frac{\partial P_i}{\partial V_j}$$

V_j MASSOUD Design Variables

P_i User-Defined Design Variables

$$\begin{bmatrix} \frac{\partial P_1}{\partial V_1} & \frac{\partial P_2}{\partial V_1} & \dots & \frac{\partial P_{fmax}}{\partial V_1} \\ \frac{\partial P_1}{\partial V_2} & \frac{\partial P_2}{\partial V_2} & \dots & \frac{\partial P_{fmax}}{\partial V_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_1}{\partial V_{jmax}} & \frac{\partial P_2}{\partial V_{jmax}} & \dots & \frac{\partial P_{fmax}}{\partial V_{jmax}} \end{bmatrix}$$

$$P_1 = V_{10} - V_1 \quad (\text{Chord})$$

$$P_2 = (V_{10} + V_1) / 2 \quad (\text{Mid-Chord Location})$$

$$P_3 = V_2 = V_{11}$$

	P_1	P_2	P_3
V_1	-1	0.5	0
V_2	0	0	1
V_{10}	1	0.5	0
V_{11}	0	0	1

M6.usd

```
# 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)
52 3 4
d 1d 2d 3d
1 -1 0.5 0
2 0 0 1
10 1 0.5 0
11 0 0 1
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



24

MASSOUD Pros and Cons

Pros

- Consistent Meshes
- No need for mesh generation
- Easy to setup (hours)
- Parameterization is fast
- Analytic sensitivity
- Compact set of DVs
- Suitable for high- and low-fidelity application

Cons

- Limited to small shape changes
- Fixed topology
- No built-in geometry constraints
- No direct CAD connection



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



25

BandAids

- Aerodynamic Shape Parameterization based on Free-Form Deformation
- General application based on free-form deformation
 - Handles complex shapes
 - DVs are not classic aerodynamic parameters



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



26

BandAids Key Ideas

1. Parameterize surface mesh
 - Avoids mesh regeneration
2. Use FFD to represent shape perturbations
 - Automates surface parameterization
3. Parameterize changes in shape perturbation, not the shape itself
 - Reduces the number of design variables



<http://fun3d.larc.nasa.gov>

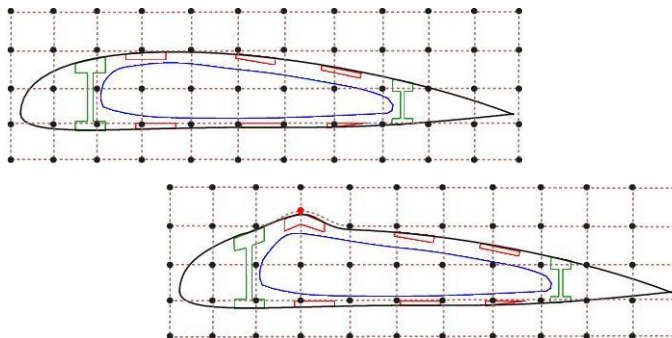
FUN3D Training Workshop
June 20-21, 2015



27

BandAids FFD (1 of 3)

- Based on algorithm used in computer animation
 - Control points are DVs
 - Immersed in Jell-O®
- Design variables have no aerodynamic significance
 - Only those near surface have significant impact



<http://fun3d.larc.nasa.gov>

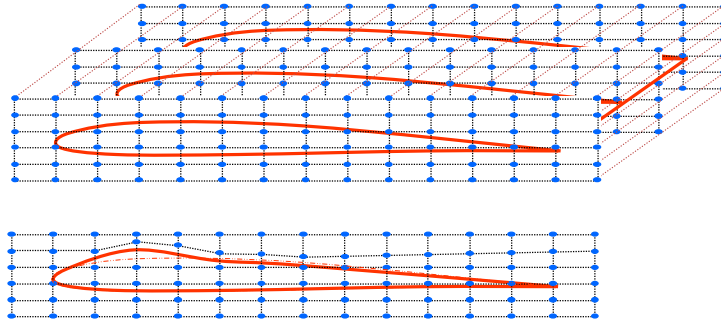
FUN3D Training Workshop
June 20-21, 2015



28

BandAids FFD (2 of 3)

- Many more control points in 3D
 - Only those near surface have impact on surface



<http://fun3d.larc.nasa.gov>

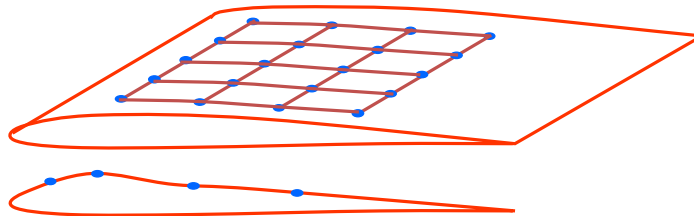
FUN3D Training Workshop
June 20-21, 2015



29

BandAids FFD (3 of 3)

- Equivalent 3D bi-variant form of tri-variant FFD
 - Collapse CPs onto surface
 - Move CP moves surface underneath
 - Number of DVs reduced from N^3 to N^2
 - 4 sided Bandaidd marking surface over geometry
 - Moves only surface to which it is collapsed
 - No MDO



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



30

BandAids Parameterizes Changes

- Shape changes are small
 - Can be represented with fewer CPs than surface
- Maintains surface mesh character/quality

$$r_n(v) = r_n^b + \Delta r_n(v)$$

Surface mesh point $r_n(v)$
 Design variable vector v
 Baseline surface mesh r_n^b
 Shape changes $\Delta r_n(v)$

NURBS control points for camber & thickness



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



31

BandAids Installation

- Distributed as source code
 - Single **Makefile** uses GNU C compiler (**gcc**)
 - Any localization must be done manually
 - Creates a single executable
 - **bandAids** parameterization and deformation



<http://fun3d.larc.nasa.gov>

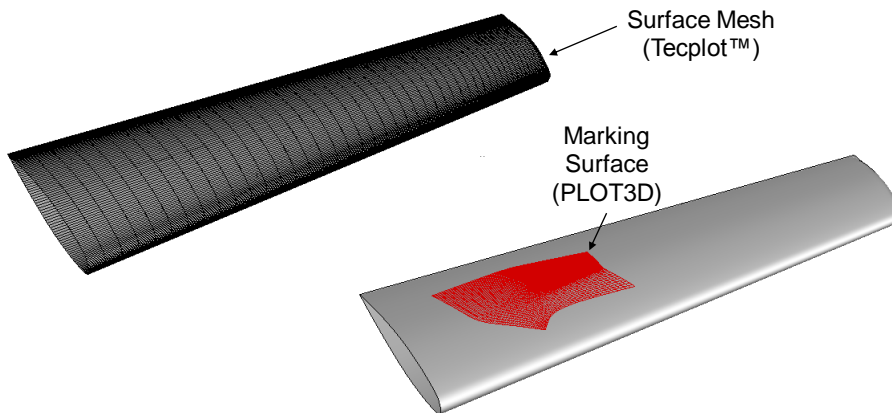
FUN3D Training Workshop
June 20-21, 2015



32

BandAids Marking Surfaces (1 of 2)

- Create structured marking surface
 - Marks portion of geometry to parameterize
 - Can span multiple geometry surfaces



<http://fun3d.larc.nasa.gov>

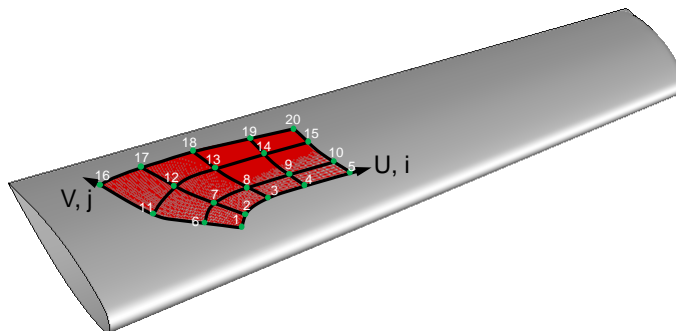
FUN3D Training Workshop
June 20-21, 2015



33

BandAids Marking Surfaces (2 of 2)

- Marking surface interpolated by reference with $n \times m$ CPs
 - $n \times m$ DVs



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



34

BandAids Execution

```
% bandAids inMesh.plt \
           inDesignSurf.p3d \
           output \
           numDesignInU \
           numDesignInV \
           [tol]
```

- “**inMesh.plt**” target mesh in Tecplot™ format
- “**inDesignSurf.p3d**” marking surface in PLOT3D format
- “**outfile**” output file name prefix
- “**numDesignInU**” number of design variables in U-direction
- “**numDesignInV**” number of design variables in V-direction
- “**tol**” optional, max gap between mesh and marking surface
- User defined variables are created if a “**bandaids.usd**” file exists at execution



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



35

BandAids Output

- Execution produces seven files:
 - “**output.bandaid**”
 - All non-zero shape information
 - Read directly by FUN3D
 - “**output.distance.plt**”
 - Tecplot™ file with the surface mesh including the distance between the surface mesh and marking surface
 - “**output.distanceSD.plt**”
 - Tecplot™ file containing surface mesh and sensitivity data
 - “**bandAidsSample.dvs**”
 - Template for input design variable file
 - “**bandAidsAll.usd**”, “**bandAidsCol.usd**”, and “**bandAidsRow.usd**”
 - Templates to base “**bandaids.usd**” used for DV linking
 - Requires a subsequent “**bandaids**” run for linked variables



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



36

BandAids Deformation

- Not necessary with FUN3D as all deformation is linear
 - Useful for validation
- Execute **bandAids** with **-deformMesh**

```
% bandAids -deformMesh \
               output.distanceSD.plt \
               my.dvs \
               new.plt
```
- “**output.distanceSD.plt**”
 - Tecplot™ file containing surface mesh and sensitivity data
- “**my.dvs**”
 - Input DV perturbations
- “**new.plt**”
 - Deformed surface mesh



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



37

BandAids Results

- Visual inspection
 - Tecplot™
 - “**output.distanceSD.plt**” contains mesh and SDs
 - (e.g. XD1, YD1, ZD1... XDndv, YDndv, ZDndv)
 - GridTool


```
% GridTool -d output.distanceSD.plt
```

 - Sliders to interactively perturb DVs



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



38

BandAids Pros and Cons

Pros

- General Application
- Consistent Meshes
- No need for mesh generation
- Easy to setup (hours)
- Parameterization is fast
- Analytic sensitivity
- Compact set of DVs
- Suitable for high- and low- fidelity application

Cons

- Non-intuitive DVs
- Limited to small shape changes
- No built-in geometry constraints
- No direct CAD connection



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



39

What We Learned

- MASSOUD parameterizes with aerodynamic parameters
 - Best applied to aerodynamic shapes
- BandAids provides general application
 - Albeit w/o intuitive parameters
- Both mesh based parameterization
- Both tools parameterize shape changes not shape
 - Reduces number of DVs
- Both provide mesh perturbation with SDs suitable for FUN3D



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



40

FUN3D v12.7 Training

Session 9: Adjoint-Based Design for Steady Flows

Eric Nielsen



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

What is an Adjoint?

Combine cost function with Lagrange multipliers Λ :

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

f = Cost function (lift/drag/boom/etc) Λ_f = Flowfield adjoint variable
 \mathbf{K} = Mesh movement elasticity matrix Λ_g = Grid adjoint variable

Differentiate with respect to \mathbf{D} :

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

$$\longrightarrow \left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}} \right]^T \Lambda_f = - \frac{\partial f}{\partial \mathbf{Q}}$$

Governing Eqns Engineering Output

This adjoint equation for the flowfield has powerful implications for:

- Error estimation & mesh adaptation
- Sensitivity analysis



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

Adjoint for Error Estimation and Mesh Adaptation

It is apparent that:

$$\Lambda_f \equiv \frac{\partial f}{\partial \mathbf{R}}$$



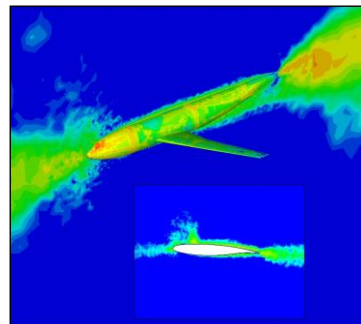
Direct relationship between local equation error and the output we are interested in

- 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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

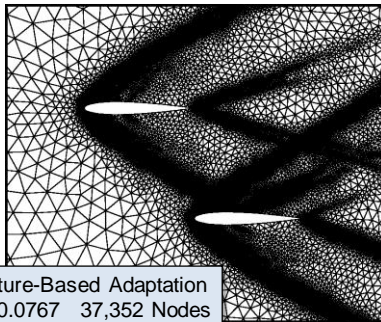
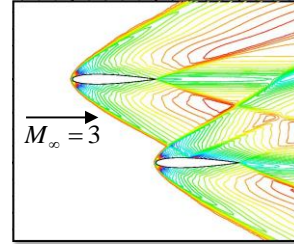


8

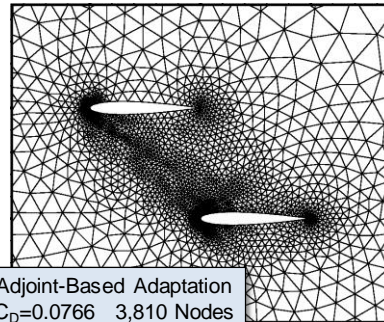
Supersonic Adjoint-Based Mesh Adaptation

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 90% smaller grid



Feature-Based Adaptation
 $C_D=0.0767$ 37,352 Nodes



Adjoint-Based Adaptation
 $C_D=0.0766$ 3,810 Nodes



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

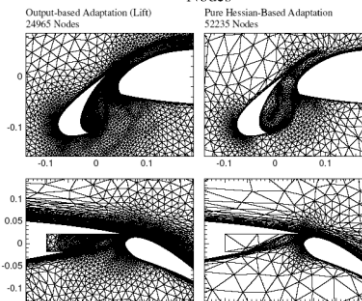
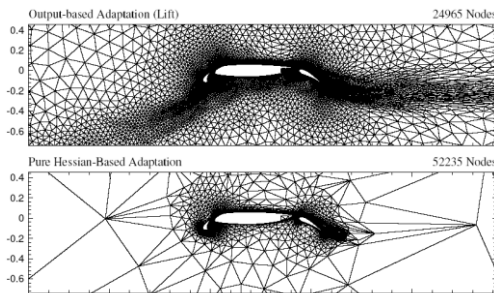
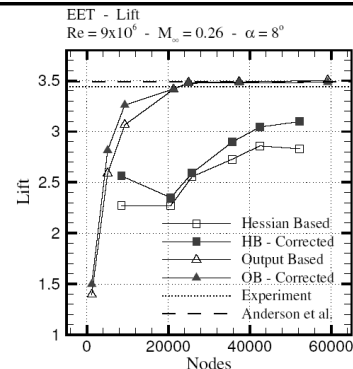


9

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

Adjoint for Sensitivity Analysis

Examine the remaining terms in the linearization:

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

$$\longrightarrow \mathbf{K}^T \Lambda_g = - \left\{ \frac{\partial f}{\partial \mathbf{X}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{X}} \right]^T \Lambda_f \right\} \quad \text{Discrete adjoint equation for mesh movement}$$

$$\longrightarrow \frac{dL}{d\mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \Lambda_f^T \frac{\partial \mathbf{R}}{\partial \mathbf{D}} - \Lambda_g^T \left[\frac{\partial \mathbf{X}}{\partial \mathbf{D}} \right]_{surf} \quad \text{Sensitivity equation}$$

Function Evaluation

1. Compute surface mesh at current \mathbf{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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

Design Variables in FUN3D

- Global flowfield variables
 - Mach number, angle of attack
- 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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



12

Objective/Constraint Functions in FUN3D

$$f_i = \sum_{j=1}^{J_i} \omega_j (C_j - C_j^*)^{p_j} \quad \begin{array}{ll} \omega = \text{weight} & C = \text{aero coeff} \\ p = \text{power} & C^* = \text{target aero coeff} \end{array}$$

- 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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

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

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



<http://fun3d.larc.nasa.gov>

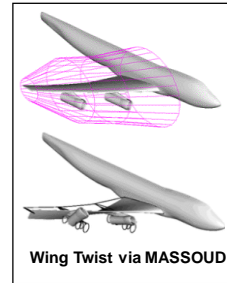
FUN3D Training Workshop
June 20-21, 2015



14

Geometry Parameterizations

- FUN3D relies on a pre-defined 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)
 - Band-aids: 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



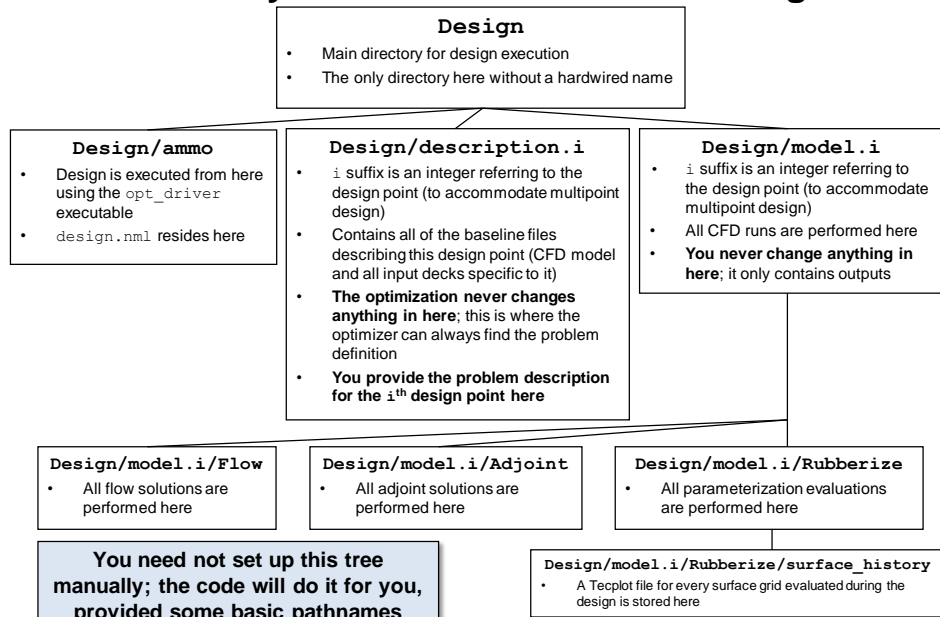
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



15

Directory Tree for FUN3D-Based Design



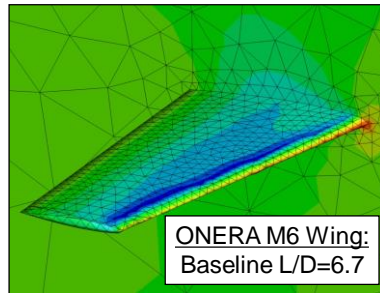
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



16

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



17

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.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



18

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



19

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



20

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



21

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



22

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



23

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



24

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
RotOrpx 0 0.000000000000000E+00 0.000000000000000E+00 0.500000000000000E+01
RotOrpy 0 0.000000000000000E+00 0.000000000000000E+00 0.500000000000000E+01
RotOrxz 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 Bandai=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
.
.
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



25

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*



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



26

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:

$$f = (L/D - 20)^2$$



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



27

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'
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



28

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)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



29

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



30

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.sdl
Iter      Natural Err Est      Error Estimate      Restarts
0         0.648914658284637E-16    0.000000000000000E+00    0

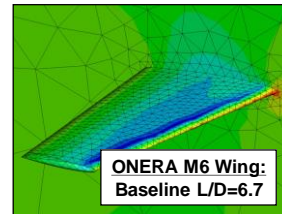
Iter      density_RMS  density_MAX  X-location  Y-location  Z-location
1         0.725550147064997E-04  0.46595E-03  0.34893E-01  0.60683E-01  0.00000E+00
Lift      0.657554528793843E-01  Drag      0.319926994134964E-01

...

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.
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



31

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



32

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.
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



33

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  137.781363854415
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`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



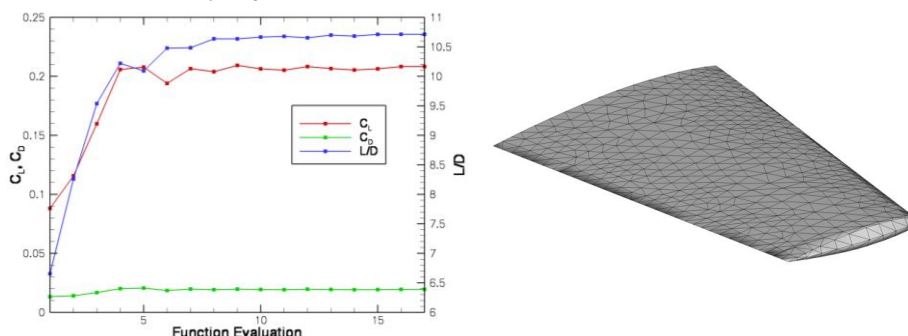
34

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.sdl.iteration.*`

Redesigned Wing:
L/D=10.7



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



35

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



36

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`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



37

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



38

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



39

FUN3D v12.7 Training

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

Mike Park



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

Learning Goals

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

Available Adaptation Modes

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

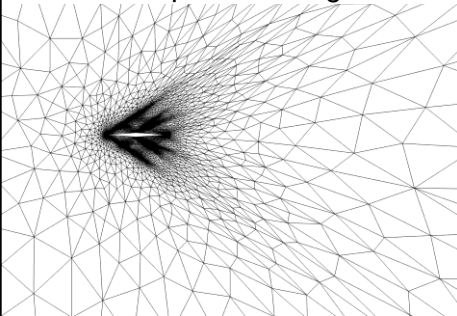


3

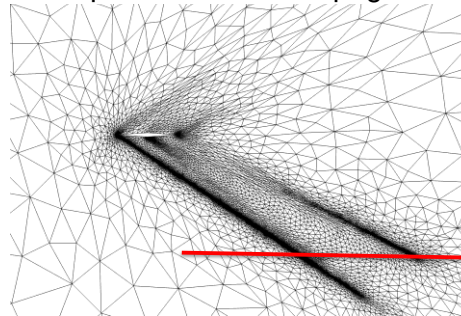
Output-Based Adaptation

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

Adapted for Drag



Adapted for Shock Propagation



<http://fun3d.larc.nasa.gov>

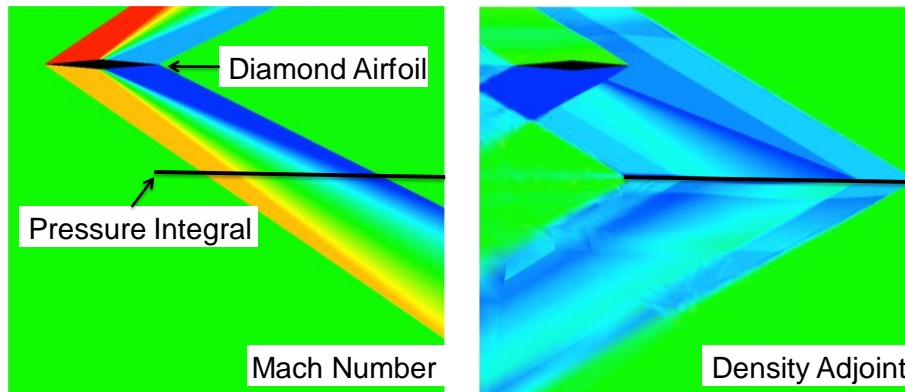
FUN3D Training Workshop
June 20-21, 2015



4

Shock Propagation Example

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

Local Error and Output Adaptation

Local error based

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

Output (adjoint) based

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



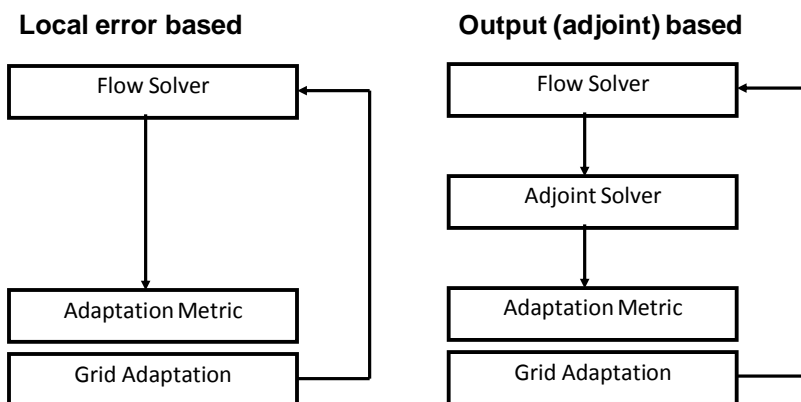
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



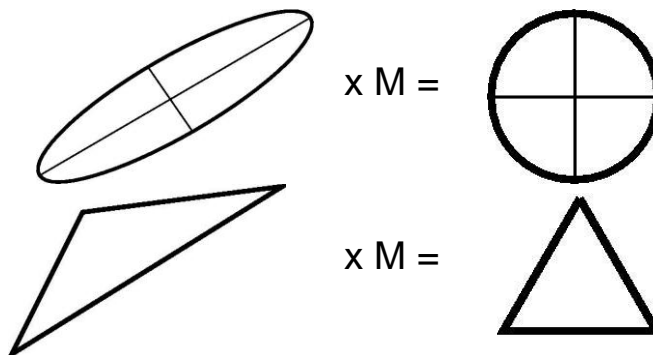
6

Adaptation Process



Metric Adaptation Mechanics

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



Metric

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

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



Metric

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



Metric Adaptation Mechanics

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

Venditti Adaptation Metric

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

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



Venditti Adaptation Metric

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

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

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



INRIA Optimal Goal-Based Metric

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



Feature Local-Error Metric

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



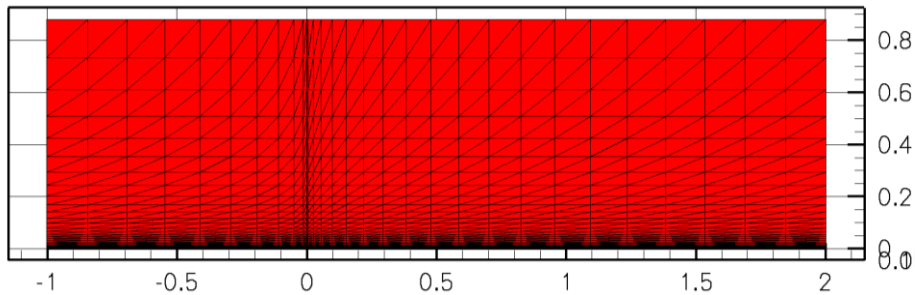
Cases

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



Supersonic Flat Plate

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



17

Initial Flow Solution

- Initial fun3d.nml grid and flow conditions

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



18

Initial Flow Solution

- Initial fun3d.nml solver parameters

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

Convergence of all residuals is critical!

Linear system Krylov projection



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



19

Initial Flow Solution

- Initial fun3d.nml co-visualization

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



20

Initial Flow Solution

- Initial fun3d.nml co-visualization

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



21

Initial Flow Solution

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

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



<http://fun3d.larc.nasa.gov>

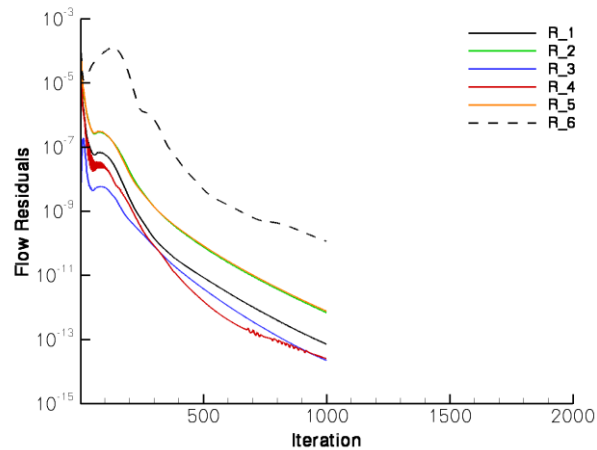
FUN3D Training Workshop
June 20-21, 2015



22

Initial Flow Solution

- Flow solver (primal) convergence history



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



23

Initial Adjoint Solution

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

```
...
Components of func 1: boundary id (0=all)/name/value/weight/target/power
0 boom_targ 0.0000000000000000 1.0 0.00000 1.000
...
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



24

Initial Adjoint Solution

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

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



25

Initial Adjoint Solution

- Initial `fun3d.nml` adjoint solver parameters

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

Typically run less adjoint iterations



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



26

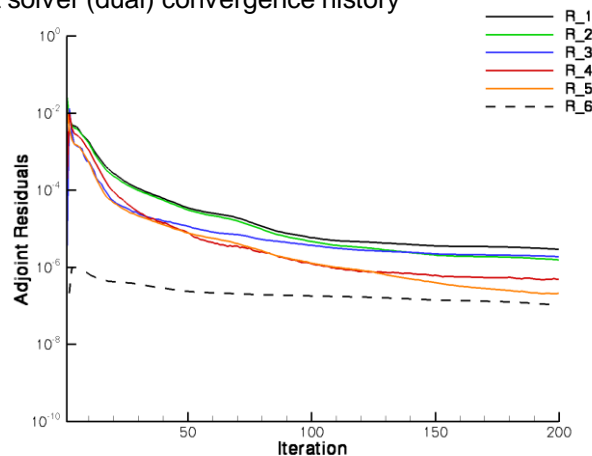
Initial Adjoint Solution

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

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

Initial Adjoint Solution

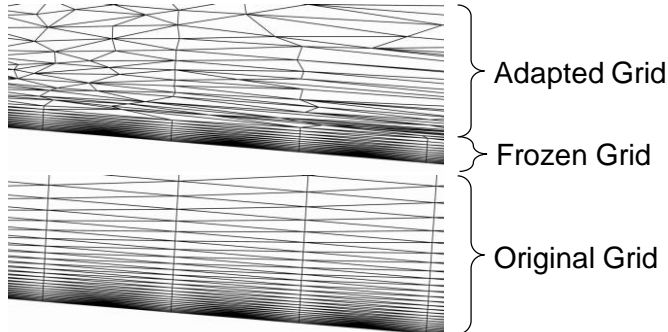
- Adjoint solver (dual) convergence history



Output-Based Adaptation

- Output-based adaptation `fun3d.nml` parameters

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



29

Output-Based Adaptation

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

7		Number of planes
1	xplane	-1.0000000000000000
2	xplane	2.0000000000000000
3	yplane	0.0000000000000000
4	yplane	0.1000000000000000
5	zplane	0.0000000000000000
6	zplane	0.0000000000000000
7	zplane	0.8813629407814508

Each plane with normal and position



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



30

Initial Adjoint Solution

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

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

`--rad` = Residual Adjoint Dot-product

`--adapt` = Activates grid adaptation

Adapted Flow Solution

- Initial `fun3d.nml` grid and flow conditions

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

New project name

New grids are always AFLR3 (ugrid) stream format

The solution is interpolated

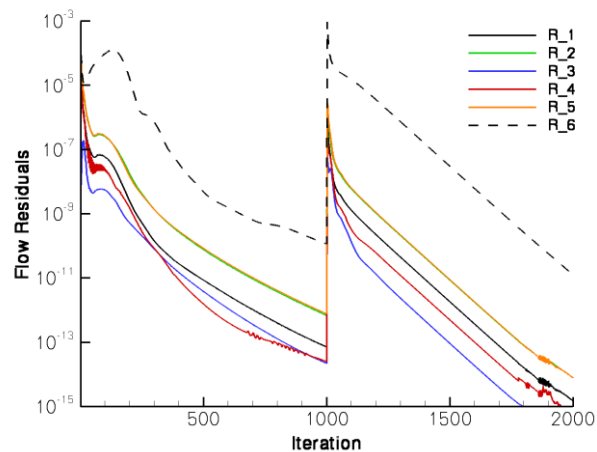
Adapted Flow Solution

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

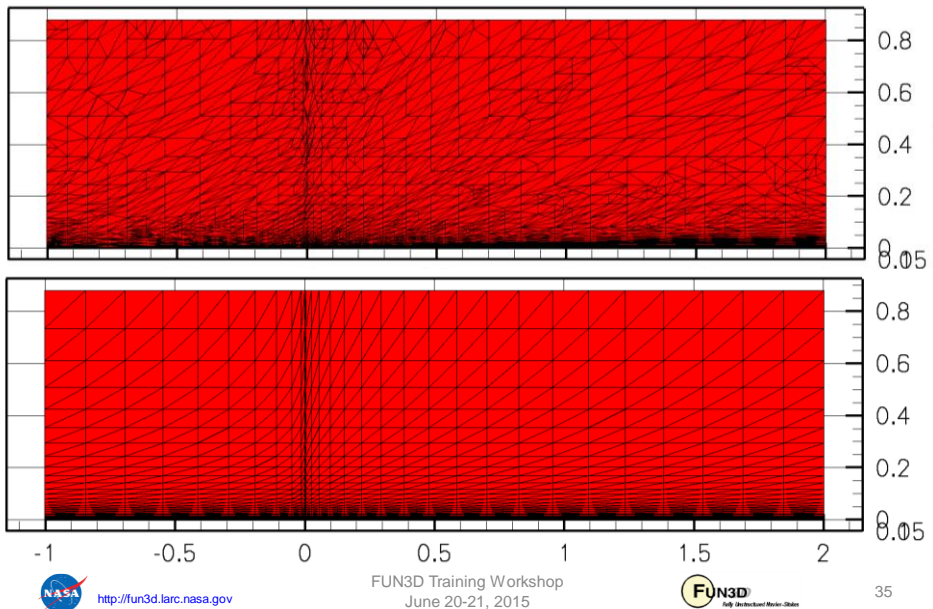
```
$ cd Flow
$ mpirun -np 8 nodet_mpi
```

Adapted Flow Solution

- Flow solver (primal) convergence history

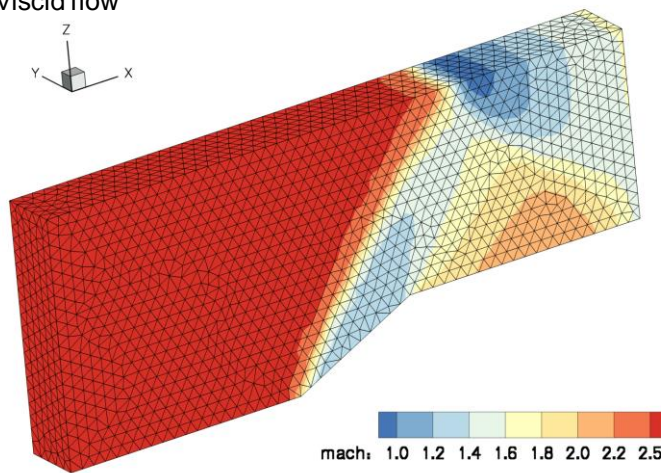


Adapted and Original Flat Plate Grid



Supersonic Ramp Feature Adaptation

- Mach 2.5, inviscid flow



Initial Flow Solution

- Initial fun3d.nml grid and flow conditions

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



37

Initial Flow Solution

- Initial fun3d.nml grid and flow conditions

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

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



38

Initial Flow Solution

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

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

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



39

Feature-Based Adaptation

- Output-based adaptation `fun3d.nml` parameters

```
&adapt_mechanics
  adapt_project = 'ramp01'           New project name
  adapt_cycles  = 10                 Passes for grid mechanics
/
&adapt_metric_construction
  adapt_feature_scalar_key = 'mach'
  adapt_feature_scalar_form = 'delta-1' Target shocks and
  adapt_output_tolerance   = 0.05      expansions
  adapt_min_edge_length    = 0.01
  adapt_max_anisotropy     = 1.0       Isotropic
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



40

Feature-Based Adaptation

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

```

8                                     Number of planes
1  zplane  0.0
2  zplane  2.0
3  yplane  0.0
4  xplane -2.0
5  yplane  0.5                                     Each plane with normal
6  xplane  3.0                                     and position
7  general_plane 0.0
   -0.5 0.0 1.0
8  zplane  0.5

```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



41

Feature-Based Adaptation

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

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

`--adapt` use adaptation mechanics



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



42

Scripting it

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

```
#!/bin/bash
```

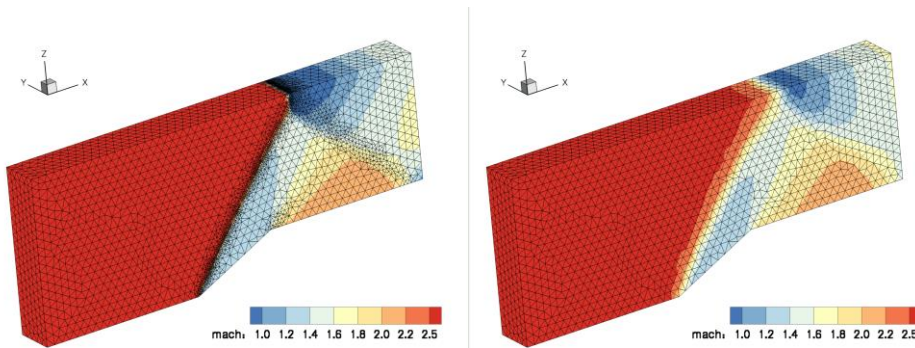
```
cp -f fun3d.nml-00 fun3d.nml
mpirun -np 8 nodet_mpi --irest 0
```

Initial solution

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

Adapt and flow solve on
new grid

Adapted and Original Grid and Mach Number



F3D script

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



<http://fun3d.larc.nasa.gov>

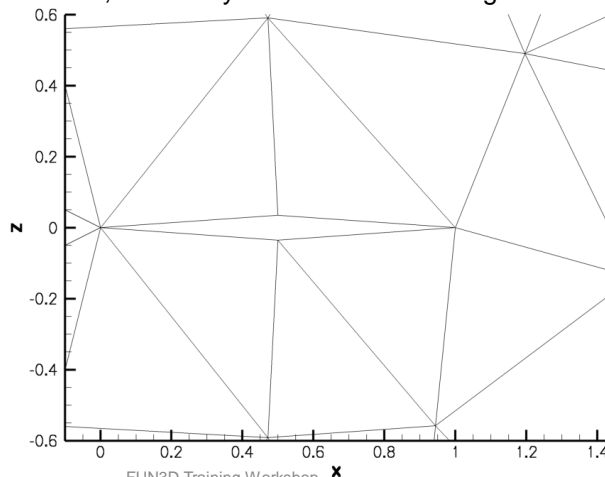
FUN3D Training Workshop
June 20-21, 2015



45

Drag-Adapted Diamond Airfoil

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

46

F3D input case_specifics example

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

```
root_project 'diamond'

number_of_processors 8

adj_cl " --outer_loop_krylov "

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

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

first_iteration 1
last_iteration 10
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



47

Namelist Setup

- Initial fun3d.nml grid and flow conditions

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



48

Namelist Setup

- Initial fun3d.nml grid and flow conditions

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



49

Namelist Setup

- Initial fun3d.nml grid and flow conditions

```
&adapt_mechanics
  adapt_library = 'refine/two'           refine version 2
  adapt_project = 'diamond02'           mechanics
/
&adapt_metric_construction
  adapt_hessian_method = 'grad'
  adapt_hessian_average_on_bound = .true.
  adapt_twod = .true.
  adapt_statistics = 'average'
  adapt_max_anisotropy = 10.0
  adapt_complexity = 1000
  adapt_gradation = 1.5
  adapt_current_h_method = 'implied'
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



50

F3D script

- Run with no subcommands for help

```
$ f3d
```

```
usage: f3d <command>
```

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



51

F3D script

- To begin and watch progress

```
$ f3d start
```

```
$ f3d watch
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

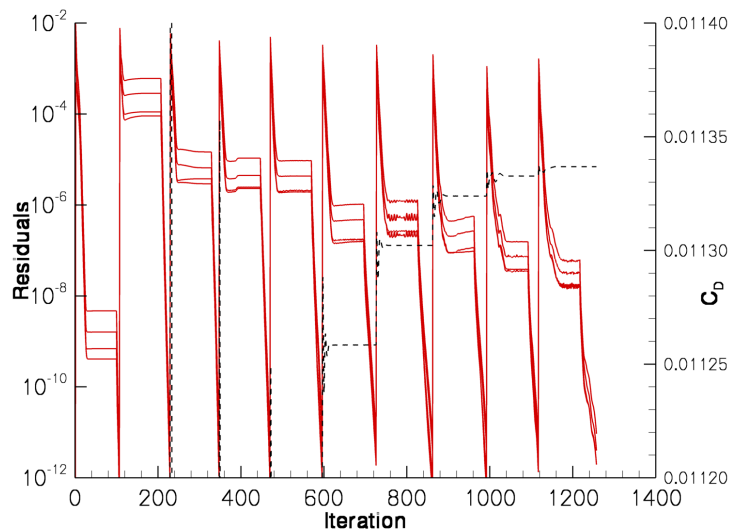


52

F3D script

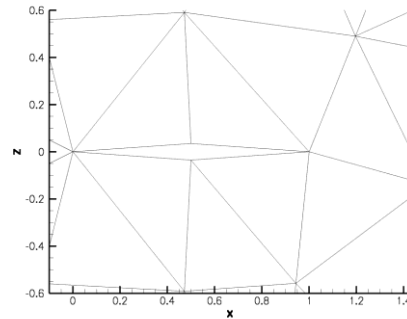
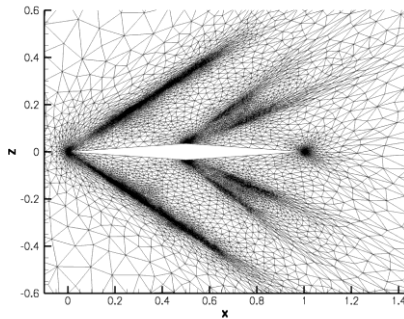
- Copies `fun3d.nml` into Flow directory and modifies it to set `project_rootname`, `restart_read`, and other options with the `nl_flo`, `nl_adj`, `nl_rad` hashes
- Backup copies of `fun3d.nml` are saved as `[project]_flow_fun3d.nml`, `[project]_dual_fun3d.nml`, and `[project]_rad_fun3d.nml`
- Backup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

Drag-Adapted Diamond Airfoil



Drag-Adapted Diamond Airfoil

- Mach 2.0, inviscid flow



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



55

Running with PBS

- Creates and submits a PBS batch script

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



56

Running with PBS

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

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



57

What Can Go Wrong?

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

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



58

What Can Go Wrong?

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



59

Evolving Process

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



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



60

FUN3D v12.7 Training

Session 11: Time-Dependent Simulations

Bob Biedron



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

Session Scope

- What this will cover
 - How to set up and run time-accurate simulations on static meshes
 - Subiteration convergence: what to strive for and why
 - Nondimensionalization
 - Choosing the time step
 - Input / Output
- What will not be covered
 - Moving-mesh, aeroelastics (covered in follow-on sessions)
- What should you already be familiar with
 - Basic steady-state solver operation and control
 - Basic flow visualization



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

Introduction

- Background
 - Many of problems of interest involve unsteady flows and may also involve moving geometries
 - Governing equations written in Arbitrary Lagrangian-Eulerian (ALE) form to account for grid speed
 - Nondimensionalization often more involved/confusing/critical
- Compatibility
 - Compressible/incompressible paths
 - Mixed elements; 2D/3D
 - Dynamic grids
 - Not compatible with generic gas model
- Status
 - Incompressible path exercised very infrequently for unsteady flows



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

Governing Equations

- Arbitrary Lagrangian-Eulerian (ALE) Formulation

$$\frac{\partial(\bar{Q}V)}{\partial t} = -\oint_{\partial V} (\bar{F} - \bar{q}\bar{W}^T) \cdot \bar{n} dS - \oint_{\partial V} \bar{F}_v \cdot \bar{n} dS = \bar{R} \quad \bar{Q} = \frac{\oint_V \bar{q} dV}{V}$$

\bar{W} = Arbitrary control surface velocity; Lagrangian if $\bar{W} = (u, v, w)^T$ (moves with fluid); Eulerian if $\bar{W} = 0$ (fixed in space)

- Discretize using N^{th} order backward differences in time, linearize \bar{R} about time level $n+1$, and introduce a pseudo-time term:

$$\left[\left(\frac{V^{n+1}}{\Delta \tau} + \frac{V^{n+1} \phi_{n+1}}{\Delta t} \right) \bar{I} - \frac{\partial \bar{R}^{n+1,m}}{\partial \bar{Q}} \right] \Delta \bar{Q}^{n+1,m} = \bar{R}^{n+1,m} - \frac{V^{n+1} \phi_{n+1}}{\Delta t} (\bar{Q}^{n+1,m} - \bar{Q}^n) - \dots + \bar{R}_{GCL}^{n+1} \\ = \bar{R}^{n+1,m} + O(\Delta t^N)$$

- Physical time-level t^n ; Pseudo-time level τ^m
- Want to drive **subiteration residual** $\bar{R}^{n+1,m} \rightarrow 0$ using pseudo-time subiterations at each time step – more later – otherwise you have more error than the expected $O(\Delta t^N)$ truncation error



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

Time Advancement - Namelist Input

- The `&nonlinear_solver_parameters` namelist in the `fun3d.nml` file governs how the solution is advanced in time
- Relevant entries - *default values shown* - some definitely need changing:

```
&nonlinear_solver_parameters
  time_accuracy      = 'steady' (i.e. not time accurate)
  time_step_nondim   = 0.0
  subiterations      = 0
  schedule_iteration  = 1      50
  schedule_cfl        = 200.0 200.0
  schedule_cfl_turb   = 50.0  50.0
  pseudo_time_stepping = "on"
  temporal_err_control = .false.
  temporal_err_floor   = 0.1
/
```

- Let's look at these in some detail (defer `time_step_nondim` to last)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

Time Advancement - Order of Accuracy

- Currently have several types of backward difference formulae (BDF) that are controlled by the `time_accuracy` component:
 - In order of formal accuracy: BDF1 (**1storder**), BDF2 (**2ndorder**), BDF2_{OPT} (**2ndorderOPT**), BDF3 (**3rdorder**), MEBDF4 (**4thorderMEBDF4**)
 - Can pretty much ignore all but BDF2_{OPT} and BDF2
 - BDF1 is least accurate; little gain in CPU time / step over 2nd order; for moving grids can be helpful to start out with BDF1 (rare)
 - BDF3 not guaranteed to be stable; feeling lucky?
 - MEBDF4 only efficient if working to very high levels of accuracy - *including spatial accuracy* - generally **not** for practical problems
 - BDF2_{OPT} (*recommended*) is a stable blend of BDF2 and BDF3 schemes; formally 2nd order accurate but error is ~1/2 that of BDF2; also allows for a more accurate estimate of the temporal error for the error controller (p.8)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

Time Advancement - Subiterations (1/4)

- Can think of each time step as a mini steady-state problem
- Subiterations (`subiterations > 0`) are essential
 - Subiteration control in *each time step* operates exactly like iteration control in a steady state case:
 - CFL ramping is available for mean flow and turbulence model – however, be aware that ramping schedule should be `< subiterations` or the specified final CFL won't be obtained
 - *We almost never ramp CFL for time-accurate cases*
 - If used, CFL ramping starts over each time step
 - Caution: the *spatial* accuracy flag, `first_order_iterations`, starts over each time step, so make sure you don't have this on
- Pseudo-time term helpful for large time steps
 - We *always* use it in our applications
 - `pseudo_time_stepping = "on"` (default)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

Time Advancement - Subiterations (2/4)

- How many subiterations?
 - In theory, should drive subiteration residual “to zero” each time step – but you cannot afford to do that
 - Otherwise have additional errors other than $O(\Delta t^2)$ (if 2nd order time)
- In a perfect world, the answer is to use the **temporal error controller**
 - `temporal_err_control = .true.`
 - `temporal_err_floor = 0.1` => iterate until the subiteration residual is 1 order lower than the (estimated) temporal error (0.01 => 2)
 - Subiterations kick out when this level of convergence is reached OR subiteration counter `> subiterations`
 - (empirically) 1 order is about the minimum; 2 orders is better, BUT...
 - Often, either the turbulence residual converges slowly or the mean flow does, and the max subiterations you specify will be reached
 - When it kicks in, the temporal error controller is the best approach, and the most efficient; even if it doesn't kick in, it can be informative



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

Time Advancement - Subiterations (3/4)

- Be wary reaching conclusions about the effect of time-step refinement unless the subiterations are “sufficiently” converged for each size step
- How to monitor and assess the subiteration convergence:
 - Printed to the screen, so you can “eyeball” it
 - With temporal error controller, if the requested tolerance is not met, message(s) will be output to the screen:
 - **WARNING: mean flow subiterations failed to converge to specified temporal_err_floor level**
 - **WARNING: turb flow subiterations failed to converge to specified temporal_err_floor level**
 - Note: when starting unsteady mode, first timestep *never* achieves target error (no error estimate first step, so target is 0)
 - Note: x-momentum residual (**R_2**) is the mean-flow residual targeted by the error controller
 - Plot it (usually best)



<http://fun3d.larc.nasa.gov>

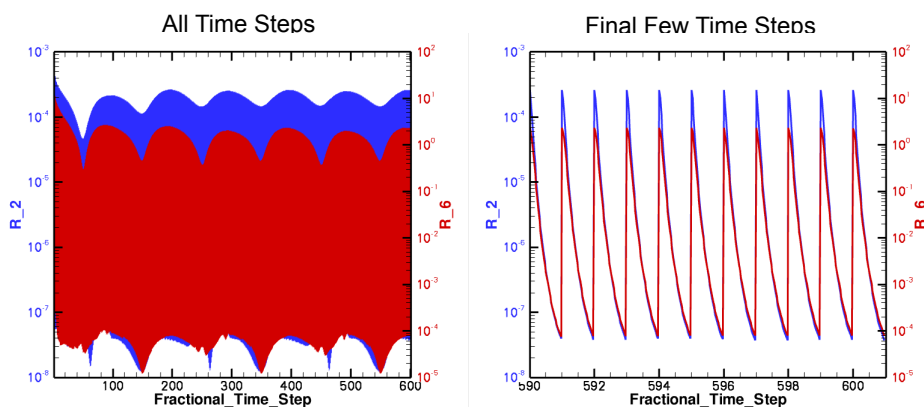
FUN3D Training Workshop
June 20-21, 2015



9

Time Advancement - Subiterations (4/4)

- Tecplot file (ASCII) with subiteration convergence history is output to a file: **[project]_subhist.dat**
 - Plot (on log scale) **R_2** (etc) vs **Fractional_Time_Step**
 - Also contains **Cl**, **Cd**, **Cm** to assess force convergence in a time step



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

Nondimensionalization of Time

- Notation: * indicates a dimensional variable, otherwise nondimensional; the reference flow state is usually free stream ("∞"), but need not be
- Define:
 - L_{ref}^* = reference length of the physical problem (e.g. chord in ft)
 - L_{ref} = corresponding length in your grid (considered *nondimensional*)
 - a_{ref}^* = reference speed of sound (e.g. ft/sec) (compressible)
 - U_{ref}^* = reference velocity (e.g. ft/sec; compressible: $U_{ref}^* = \text{Mach } a_{ref}^*$)
 - t^* = time (e.g. sec)
- Then nondimensional time in FUN3D is related to physical time by:
 - $t = t^* a_{ref}^* (L_{ref}/L_{ref}^*)$ (compressible)
 - $t = t^* U_{ref}^* (L_{ref}/L_{ref}^*)$ (incompressible)
 - *Usually* have $L_{ref}/L_{ref}^* = 1^*$, but need not - e.g. typical 2D airfoil grid
 - L_{ref}/L_{ref}^* appears because Re in FUN3D is input *per unit grid length*



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

Determining the Time Step

- Identify a **characteristic time** t_{chr}^* that you need to resolve with some level of accuracy in your simulation; perhaps:
 - Some important shedding frequency f_{shed}^* (Hz) is known or estimated
 $t_{chr}^* \sim 1 / f_{shed}^*$
 - Periodic motion of the body $t_{chr}^* \sim 1 / f_{motion}^*$
 - A range of frequencies in a DES-type simulation $t_{chr}^* \sim 1 / f_{highest}^*$
 - If none of the above, you can estimate the time it takes for a fluid particle to cross the characteristic length of the body, $t_{chr}^* \sim L_{ref}^* / U_{ref}^*$
 - $t_{chr} = t_{chr}^* a_{ref}^* (L_{ref}/L_{ref}^*)$ (comp) $t_{chr} = t_{chr}^* U_{ref}^* (L_{ref}/L_{ref}^*)$ (incomp)
- Say you want N time steps within the characteristic time:
 - $\Delta t = t_{chr} / N = \text{time_step_nondim}$
- Figure an absolute *minimum* of $N = 100$ for reasonable resolution of t_{chr} with a 2nd order scheme - really problem dependent (*frequencies > f^* may be important*); but don't over resolve time if space is not well resolved too



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



12

Tutorial Case: Unsteady Flow, High AoA (1/7)

- Test case located in: tutorials/flow_unsteady_airfoil_high_AoA
 - `run_tutorial.sh` script starts with a 2000 time step restart file, runs an additional 100 steps, and makes plots that follow
- Consider flow past a (2D) NACA 0012 airfoil at 45° angle of attack - the flow separates and is unsteady
 - $Re_{c^*} = 4.8$ million, $M_{ref} = 0.6$, assume $a^*_{ref} = 340$ m/s
 - chord = 0.1m, chord-in-grid = 1.0 so $L_{ref}/L^*_{ref} = 1.0/0.1 = 10$ (m^{-1})
 - Say we know from experiment that lift oscillations occur at ~450 Hz
 - $t^*_{chr} = 1 / f^*_{chr} = 1 / 450$ Hz = 0.002222 s
 - $t_{chr} = t^*_{chr} a^*_{ref} (L_{ref}/L^*_{ref}) = (0.002222)(340)(10) = 7.555$
 - $\Delta t = t_{chr} / N$ so $\Delta t = 0.07555$ for 100 steps / lift cycle
 - By way of comparison, for $M = 0.6$, $a^*_{ref} = 340$ m/s, and $L^*_{ref} = 0.1$ m it takes a fluid particle $\sim (0.1)/(204) = 0.00049$ s to pass by the airfoil; this leads to smaller, more conservative estimate for the time step, by about a factor of 4



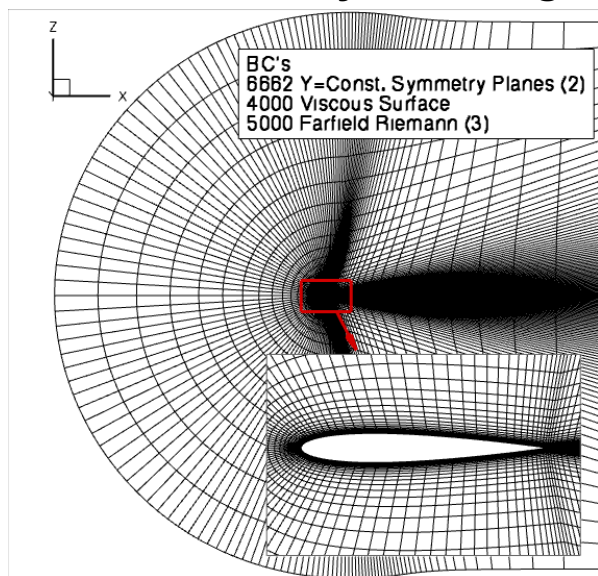
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

Tutorial Case: Unsteady Flow, High AoA (2/7)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

Tutorial Case: Unsteady Flow, High AoA (3/7)

- Flow viz: output u-velocity and y-component of vorticity
- Relevant fun3d.nml namelist data (note: many defaults assumed)

```
&project
  project_rootname = "n0012_i153"
  case_title = "NACA 0012 airfoil, 2D Hex Mesh"
/
&global
  boundary_animation_freq = 5
/
&raw_grid
  grid_format = "aflr3"
  data_format = "ASCII"
  twod_mode = .true.
/
&reference_physical_properties
  mach_number = 0.60
  reynolds_number = 4800000.00
  temperature = 520.00
  temperature_units = 'Rankine'
  angle_of_attack = 45.0
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



15

Tutorial Case: Unsteady Flow, High AoA (4/7)

- Relevant fun3d.nml namelist data (cont)

```
&force_moment_integ_properties
  x_moment_center = 0.25
/
&nonlinear_solver_parameters
  time_accuracy = "2ndorderOPT" ! Our Workhorse Scheme
  time_step_nondim = 0.07555 ! 100 steps/cycle @ 450 Hz
  temporal_err_control = .true. ! Enable error-based knockout
  temporal_err_floor = 0.1 ! Exit 1 order below error estimate
  subiterations = 30 ! No more than 30
  schedule_cfl = 50.00 50.00 ! constant cfl each step; no ramping
  schedule_cfl_turb = 30.00 30.00
/
&code_run_control
  steps = 100 ! need ~2000 steps to be periodic from freestream
/
&boundary_output_variables
  primitive_variables = .false. ! turn off default
  y = .false. ! So tecplot displays correct 2D orientation by default
  u = .true.
  vort_y = .true.
/
```



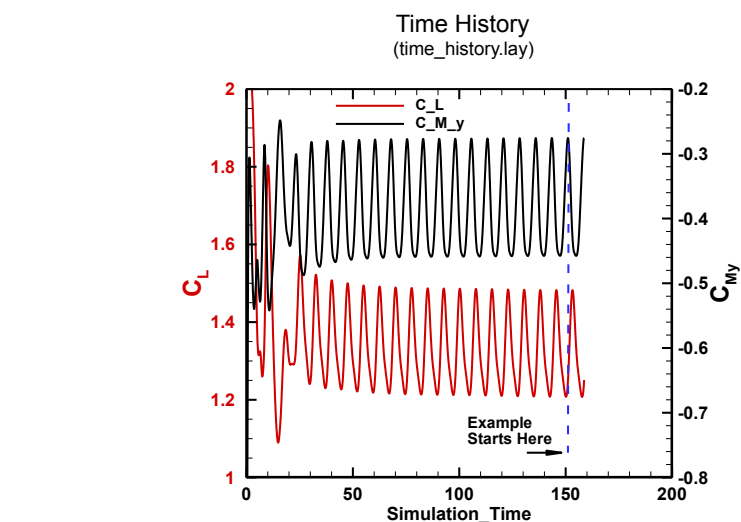
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



16

Tutorial Case: Unsteady Flow, High AoA (5/7)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

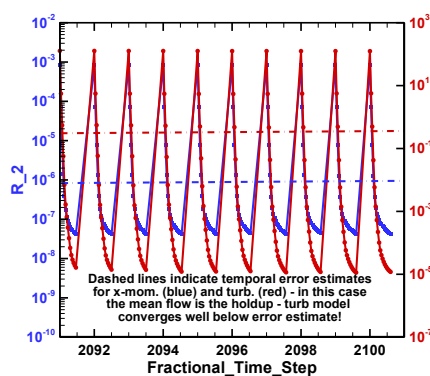


17

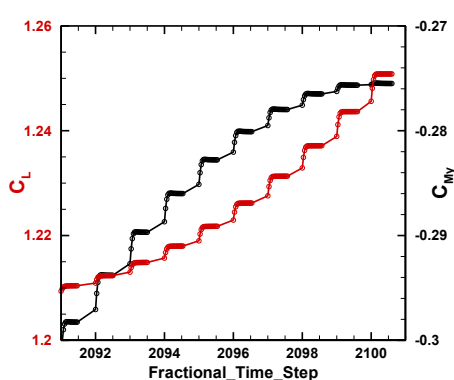
Tutorial Case: Unsteady Flow, High AoA (6/7)

- Subiterations converge? `grep "WARNING" screen_output | wc`
 - In this case, all steps converge to the specified tolerance

Subiteration Residuals, Final 10 Steps



Subiteration Lift & PM, Final 10 Steps



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015

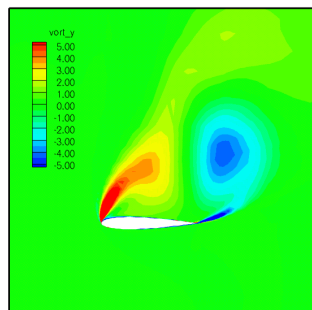
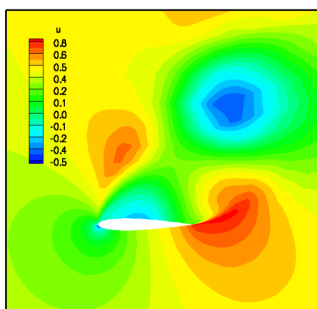


18

Tutorial Case: Unsteady Flow, High AoA (7/7)

- Animation of Results

X-Component of Velocity



Y-Component of Vorticity



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



19

List of Key Input/Output Files

- Beyond basics like `fun3d.nml`, etc.:
- Input
 - none
- Output
 - `[project]_subhist.dat`
 - Use to check subiteration residual and force/moment convergence



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



20

FUN3D v12.7 Training

Session 12: Dynamic-Grid Simulations

Bob Biedron



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



1

Session Scope

- What this will cover
 - How to set up and run time-accurate simulations on dynamic meshes
 - Nondimensionalization
 - Choosing the time step
 - Body / Mesh motion options
 - Input / Output
- What will not be covered
 - Specifics for overset and aeroelastic: covered in follow-on sessions
- What should you already be familiar with
 - Basic steady-state solver operation and control
 - Basic flow visualization



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



2

Introduction

- Background
 - Many of problems of interest involve moving or deforming geometries
 - Governing equations written in Arbitrary Lagrangian-Eulerian (ALE) form to account for grid speed
 - Nondimensionalization often more involved/confusing/critical
- Compatibility
 - Fully compatible for compressible/incompressible flows; mixed elements; 2D/3D
 - Not compatible with generic gas model
- Status
 - Compressible path with moving grids is exercised routinely; incompressible path much less so
 - 6-DOF option has had very limited testing / usage



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



3

Governing Equations

- Arbitrary Lagrangian-Eulerian (ALE) Formulation

$$\frac{\partial(\bar{Q}V)}{\partial t} = -\oint_{\partial V} (\bar{F} - \bar{q}\bar{W}^T) \cdot \bar{n} dS - \oint_{\partial V} \bar{F}_v \cdot \bar{n} dS = \bar{R} \quad \bar{Q} = \frac{\oint_V \bar{q} dV}{V}$$

\bar{W} = Arbitrary control surface velocity; Lagrangian if $\bar{W} = (u, v, w)^T$ (moves with fluid); Eulerian if $\bar{W} = 0$ (fixed in space)

- Discretize using N^{th} order backward differences in time, linearize \bar{R} about time level $n+1$, and introduce a pseudo-time term:

$$\left[\left(\frac{V^{n+1}}{\Delta \tau} + \frac{V^{n+1} \phi_{n+1}}{\Delta t} \right) \bar{I} - \frac{\partial \bar{R}^{n+1,m}}{\partial \bar{Q}} \right] \Delta \bar{Q}^{n+1,m} = \bar{R}^{n+1,m} - \frac{V^{n+1} \phi_{n+1}}{\Delta t} (\bar{Q}^{n+1,m} - \bar{Q}^n) - \dots + \bar{R}_{GCL}^{n+1} \\ = \bar{R}^{n+1,m} + O(\Delta t^N)$$

- Physical time-level t^n ; Pseudo-time level τ^m
- Need to drive **subiteration residual** $\bar{R}^{n+1,m} \rightarrow 0$ using pseudo-time subiterations at each time step – more later – otherwise you have more error than the expected $O(\Delta t^N)$ truncation error



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



4

Mesh / Body Motion (1/2)

- Motion is triggered either by setting `moving_grid = .true.` in `&global` (`fun3d.nml`), or by the command line `--moving_grid`
- All dynamic-mesh simulations require some input data via an auxiliary namelist file: `moving_body.input`
- A body is defined as a user-specified collection of solid boundaries in grid
- Body motion options:
 - Several built-in functions for rigid-body motion: translation and/or rotation with either constant velocity or periodic displacement
 - Read a series of surface files – body can be either rigid or deforming
 - Read a series of 4x4 transform matrices - rigid body
 - 6 DOF via UAB/Kestrel library “libmo”
 - Limited distribution
 - Requires configuring with `--with-sixdof=/path/to/6DOF`
 - Application-specific: mode-shape based aeroelasticity (linear structures); rotorcraft nonlinear beam



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



5

Mesh / Body Motion (2/4)

- Chose a mesh-motion option than can accommodate the desired body-motion option
- Mesh motion options:
 - Rigid - maximum 1 body containing all solid surfaces (unless overset)
 - Deforming – allows multiple bodies without overset; can be limited to relatively small displacements before mesh cells collapse
 - Combine rigid and/or deforming with overset for large displacements / multiple bodies
- Rigid mesh motion performed by application of 4x4 transform matrix to all points in the mesh - fast; positivity of cell volumes guaranteed to be maintained
 - Complex transforms can be built up from simple ones: matrix multiply
 - Allows parent-child motion (child follows parent but can have its own motion on top of that)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



6

Mesh / Body Motion (3/4)

- Mesh deformation handled via solution of a linear elasticity PDE:

$$\nabla \cdot [\mu(\nabla u + \nabla u^T) + \lambda(\nabla \cdot u)I] = f = 0$$

$$\lambda = \frac{Ev}{(1+\nu)(1-2\nu)} \quad \mu = \frac{E}{2(1+\nu)}$$

- ν (Poisson's ratio) is fixed; E (Young's modulus) is selectable as:
 - 1 / slen `--elasticity 1` (default)
 - 1 / volume `--elasticity 2` (rarely used anymore)
 - 1 / slen**2 `--elasticity 5` (last ditch for difficult problems)
- Elasticity solved via GMRES method; CPU intensive - can be 30% or more of the flow solve time; check convergence (screen output)
- Fairly robust, but *can* generate negative cell volumes; code stops
- “untangling” step attempted if neg. volumes generated – ***tet meshes only***



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



7

Mesh / Body Motion (4/4)

- GMRES solver used for mesh **deformation** has default parameter settings which can be adjusted in the namelist `&elasticity_gmres` (in the `fun3d.nml` file):

```
ileft   nsearch nrestarts   tol
      1       +50         10   1.e-06
```

- You generally won't have to adjust these values
- Exception: “structured” grids with very tight wake spacing can be very hard to deform and you may need to set `tol` very small, e.g. 1.e-12 (and will need more restarts); usually not an issue with typical grids
- If negative volumes are generated and not successfully untangled, try reducing `tol`, which in turn may require a larger value of `nrestarts`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



8

Nondimensionalization of Motion Data (1/2)

- Recall: * indicates a dimensional variable, otherwise nondimensional
- Typical motion data we need to nondimensionalize: translational velocity, translational displacement, angular velocity, and oscillation frequency
- Angular or translational displacements / velocities are input into FUN3D as magnitude and direction
- Displacement input: angular in degrees; translational $\Delta \vec{x} = \Delta \vec{x}^* / (L_{ref}^* / L_{ref})$
- Translational velocity is nondimensionalized just like flow velocity:
 - U^* = translation speed of the vehicle (e.g. ft/s)
 - $U = U^* / a_{ref}^*$ (comp.; this is a Mach No.) $U = U^* / U_{ref}^*$ (incomp)
- Rotation rate:
 - Ω^* = body rotation rate (e.g. rad/s)
 - $\Omega = \Omega^* (L_{ref}^* / L_{ref}) / a_{ref}^*$ (comp) $\Omega = \Omega^* (L_{ref}^* / L_{ref}) / U_{ref}^*$ (incomp)
 - Other variants on specified rotation rate are possible, e.g. rotor tip speed, from which $\Omega^* = U_{tip}^* / R^*$



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



9

Nondimensionalization of Motion Data (2/2)

- Oscillation frequency of the physical problem can be specified in different forms
 - f^* = frequency (e.g. Hz)
 - ω^* = circular frequency (rad/s)
 $= 2 \pi f^*$
 - k = reduced frequency, $k = \frac{1}{2} L_{ref}^* \omega^* / U_{ref}^*$ (be careful of exact definition - sometimes a factor of $\frac{1}{2}$ is not used)
- Built-in sinusoidal oscillation in FUN3D is defined as $\sin(2 \pi f t + \delta)$ where the nondimensional frequency f and phase lag δ are user-specified
- So the corresponding nondimensional frequency for FUN3D is
 - $f = f^* (L_{ref}^* / L_{ref}) / a_{ref}^*$ (comp) $f = f^* (L_{ref}^* / L_{ref}) / U_{ref}^*$ (incomp)
 - $f = \omega^* (L_{ref}^* / L_{ref}) / (2 \pi a_{ref}^*)$ $f = \omega^* (L_{ref}^* / L_{ref}) / (2 \pi U_{ref}^*)$
 - $f = k M_{ref}^* / (\pi L_{ref})$ $f = k / (\pi L_{ref})$



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



10

Overview of `moving_body.input`

- A body is defined as a collection of solid boundaries in the grid
- The specifics of body / mesh motion are set in one or more namelists that are put in a file called `moving_body.input` - this file *must* be provided when `moving_grid` is triggered (as a CLO or `&global` entry)
 - The `&body_definitions` namelist defines one or more bodies that move and is *always* needed in a dynamic-grid simulation
 - The `&forced_motion` namelist provides a limited means of defining basic translations and rotations as functions of time
 - The `&motion_from_file` namelist defines the motion of a rigid body from a sequence of 4x4 transform matrices
 - The `&surface_motion_from_file` namelist defines the motion of a rigid or deforming body from a time sequence of boundary surfaces
 - The `&observer_motion` namelist provides a means of generating boundary animation output from a non-stationary reference frame
- `&body_definitions` is required with `moving_grid`, others optional



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



11

Overview of `&body_definitions` Namelist

- Only most-used items shown here – see manual for complete list
- The `&body_definitions` namelist defines the bodies that move (defaults shown; most need changing)

```
&body_definitions          ! below, b=body i=boundary
  n_moving_bodies          = 0 ! how many bodies in motion
  body_name(b)             = '' ! must set unique name for each
  parent_name(b)           = '' ! child inherits motion of parent
  n_defining_boundary(b)= 0 ! how many boundaries define body
  defining_boundary(i,b)= 0 ! list of boundaries defining body
  motion_driver(b)         = 'none' ! mechanism driving body motion
  mesh_movement(b)         = 'static' ! specifies how mesh will move
/
```

- **Caution:** boundary numbers must reflect any lumping applied at run time!
- All variables above except `n_moving_bodies` are set for each body
- The blank string(`''`) for `parent_name` => inertial frame



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



12

Overview of &body_definitions (cont.)

- Options for `motion_driver` (default: `'none'`)
 - `'forced'`
 - Built-in forcing functions for rigid-body motion, const. or periodic
 - `'surface_file'`
 - File with surface meshes at selected times; interpolates in between
 - `'motion_file'`
 - File with 4x4 transforms at selected times; “interpolates” in between
 - `'6dof'`
 - relies on calls to “libmo” functions
 - `'aeroelastic'`
 - modal aeroelastics
 - All the above require additional namelists to specify details; next slide outlines namelist required when `motion_driver='forced'`
- Options for `mesh_movement` (default: `'static'`)
 - `'rigid'`, `'deform'`, `'rigid+deform'`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



13

Overview of &forced_motion Namelist

- Use `&forced_motion` namelist to specify a limited set of built-in motions

```
&forced_motion      ! below, index b=body#
rotate(b)           ! how to rotate this body: 0 don't (default);
                    ! 1 constant rotation rate; 2 sinusoidal in time
rotation_rate(b)    ! body rotation rate; used only if rotate = 1
rotation_freq(b)    ! frequency of oscillation; use only if rotate = 2
rotation_amplitude(b) ! oscillation amp. (degrees); only if rotate=2
rotation_vector_x(b) ! x-comp. of unit vector along rotation axis
rotation_vector_y(b) ! y-comp. of unit vector along rotation axis
rotation_vector_z(b) ! z-comp. of unit vector along rotation axis
rotation_origin_x(b) ! x-coord. of rotation center (to fix axis)
rotation_origin_y(b) ! y-coord. of rotation center
rotation_origin_z(b) ! z-coord. of rotation center
/
```

- There are analogous inputs for translation (`translation_rate`, etc.)
- See manual for complete list
- Note: FUN3D's sinusoidal oscillation function (translation or rotation) has 2π built in, e.g `sin(2 π rotation_freq t)`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



14

Output Files

- In addition to the usual output files, for forced / 6-DOF motion there are 3 ASCII Tecplot files for each body
 - **PositionBody_N.dat** tracks linear (x,y,z) and angular (yaw, pitch, roll) displacement of the “CG” (rotation center)
 - **VelocityBody_N.dat** tracks linear (V_x, V_y, V_z) and angular ($\Omega_x, \Omega_y, \Omega_z$) velocity of the “CG” (rotation center)
 - **AeroForceMomentBody_N.dat** tracks force components (F_x, F_y, F_z) and moment components (M_x, M_y, M_z)
 - Data in all files are nondimensional by default (e.g. “forces” are actually force coefficients); **moving_body.input** file has option to supply dimensional reference values such that *this* data is output in dimensional form - see manual/website for details
 - Forces are by default given in the inertial reference system; **moving_body.input** file has option to output forces in the body-fixed system - see manual/website for details



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



15

Tutorial Case: Pitching Airfoil (1/8)

- Test case located in: tutorials/flow_unsteady_airfoil_pitching
 - **run_tutorial.sh** script starts with a 600 time step restart file, runs an additional 100 steps, and makes plots that follow
- Consider one of the well known AGARD pitching airfoil experiments, “Case 1”
 - $Re_{c^*} = 4.8$ million, $M_{inf} = 0.6$, chord = $c^* = 0.1m$, chord-in-grid = 1.0
 - Reduced freq. $k = 2\pi f^* / (U_{inf}^* / 0.5c^*) = 0.0808$, ($f^* = 50.32$ Hz)
 - Angle of attack variation (exp): $\alpha = 2.89 + 2.41 \sin(2\pi f^* t^*)$ (deg)
- Setting the FUN3D data:
 - **angle_of_attack = 2.89** **rotation_amplitude = 2.41**
 - Recall $f = k M_{ref}^* / \pi$ from the 2nd nondimensionalization slide
 - **rotation_freq = f = 0.0808 (0.6) / 3.14... = 0.01543166**
 - So in this case we actually didn’t have to use any dimensional data since the exp. frequency was given as a reduced (non dim.) frequency



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



16

Tutorial Case: Pitching Airfoil (2/8)

- Setting the FUN3D data (cont):
 - Time step: the motion has gone through one cycle of motion when $t = T$, so that

$$\sin(2\pi \text{ rotation_freq } T) = \sin(2\pi)$$

$$T = 1 / \text{rotation_freq} \quad (\text{this is our } t_{\text{chr}})$$
 for N steps / cycle, $T = N \Delta t$ so

$$\Delta t = T / N = (1 / \text{rotation_freq}) / N$$
 - Take 100 steps to resolve this frequency:

$$\Delta t = (1 / 0.01543166) / 100 = 0.64801842$$
 - Alternatively, could use $t_{\text{chr}} = (1 / f^*) a^*_{\text{inf}} (L_{\text{ref}} / L^*_{\text{ref}})$, with $f^* = 50.32$ Hz, and assume value for a^*_{inf}



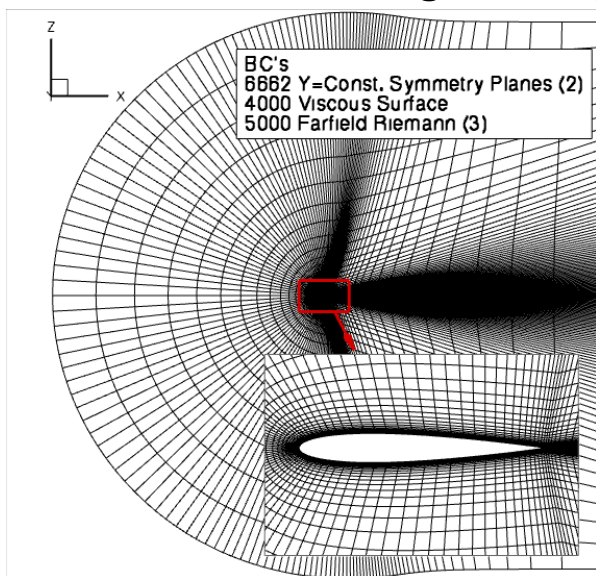
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



17

Tutorial Case: Pitching Airfoil (3/8)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



18

Tutorial Case: Pitching Airfoil (4/8)

- Relevant `fun3d.nml` data

```
&global
  moving_grid = .true.
/
&nonlinear_solver_parameters
  temporal_err_control = .true.      ! Turn on
  temporal_err_floor   = 0.1         ! Exit 1 order below estimate
  time_accuracy        = "2ndorderOPT" ! Our Workhorse Scheme
  time_step_nondim     = 0.64801842 ! 100 steps/pitch cycle
  subiterations        = 30
  schedule_cfl         = 50.00 50.00 ! constant cfl each step
  schedule_cfl_turb    = 30.00 30.00
/
```

- Relevant `moving_grid.input` data

```
&body_definitions
  n_moving_bodies      = 1,          ! number of bodies
  body_name(1)         = 'airfoil',  ! name must be in quotes
  n_defining_bndry(1)  = 1,          ! one boundary defines the airfoil
  defining_bndry(1,1)  = 5,          ! (boundary, body)
  motion_driver(1)     = 'forced'
  mesh_movement(1)     = 'rigid',
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



19

Tutorial Case: Pitching Airfoil (5/8)

- Relevant `moving_grid.input` data (cont)

```
&forced_motion
  rotate(1)            = 2,          ! type: sinusoidal
  rotation_freq(1)     = 0.01543166, ! reduced rotation frequency
  rotation_amplitude(1) = 2.41,      ! pitching amplitude
  rotation_origin_x(1) = 0.25,       ! x-coordinate of rotation origin
  rotation_origin_y(1) = 0.0,        ! y-coordinate of rotation origin
  rotation_origin_z(1) = 0.0,        ! z-coordinate of rotation origin
  rotation_vector_x(1) = 0.0,        ! unit vector x-component along
                                     ! rotation axis
  rotation_vector_y(1) = 1.0,        ! unit vector y-component along
                                     ! rotation axis
  rotation_vector_z(1) = 0.0,        ! unit vector z-component along
                                     ! rotation axis
/
```



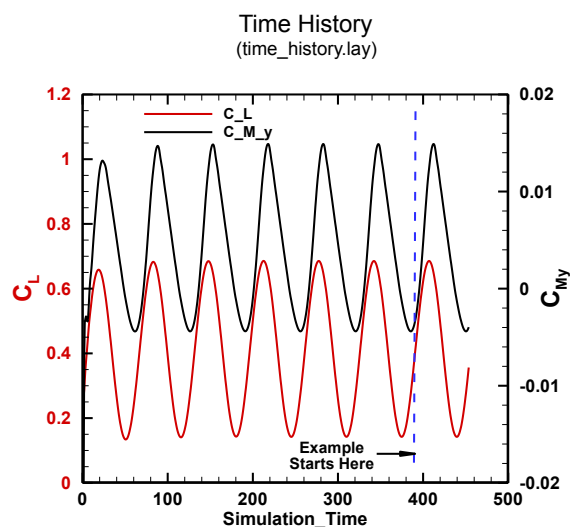
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



20

Tutorial Case: Pitching Airfoil (6/8)



<http://fun3d.larc.nasa.gov>

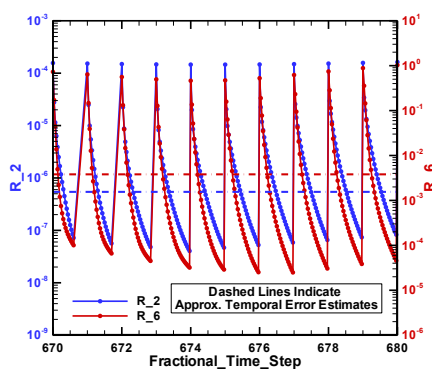
FUN3D Training Workshop
June 20-21, 2015



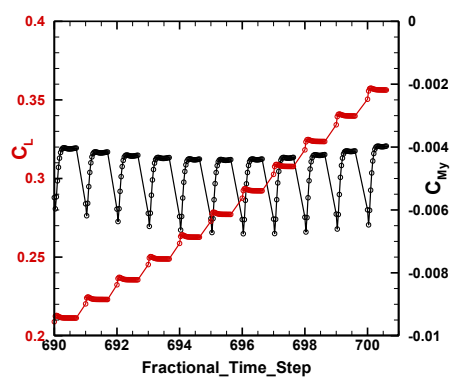
21

Tutorial Case: Pitching Airfoil (6/8)

Subiteration Residuals, Final 10 Steps
(mean flow just misses tolerance)
(subit_history.lay)



Subiteration Lift & PM, Final 10 Steps
(subit_force_history.lay)



<http://fun3d.larc.nasa.gov>

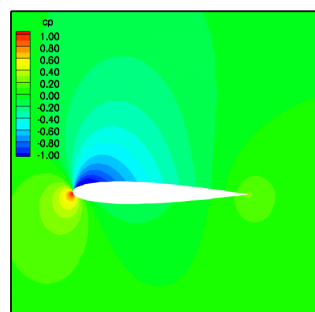
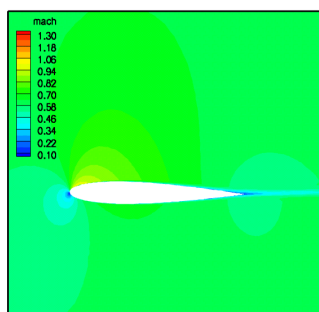
FUN3D Training Workshop
June 20-21, 2015



22

Tutorial Case: Pitching Airfoil (7/8)

Mach Number
(mach_animation.lay)



Pressure Coefficient
(cp_animation.lay)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



23

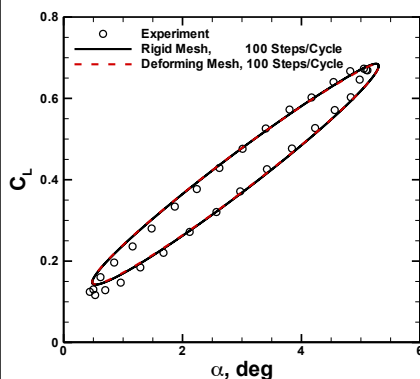
Tutorial Case: Pitching Airfoil (8/8)

Comparison with Landon, AGARD-R-702, Test Data, 1982

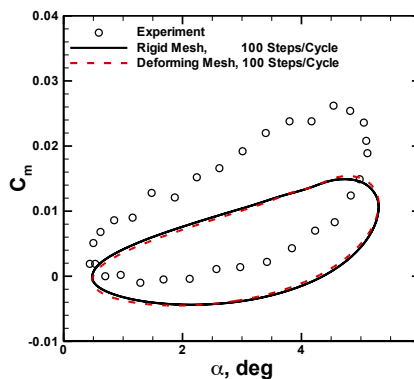
Note: comparison typical of other published CFD results

These plots not generated as part of the tutorial

Lift vs. Alpha



Pitching Moment vs. Alpha



Rigid mesh and deforming mesh produce nearly identical results



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



24

Troubleshooting Body / Grid Motion

- When first setting up a dynamic mesh problem, suggest using either the following in the `&global` namelist
 - `body_motion_only = .true.`
 - `grid_motion_only = .true.`
- Both options turn off the flow solution for faster processing (memory footprint is the same however)
 - `body_motion_only` especially useful for 1st check of a deforming mesh case since the elasticity solver is also bypassed
 - `grid_motion_only` performs all mesh motion, including elasticity solution – in a deforming case this can tell you up front if negative volumes will be encountered
 - Caveat: can't really do this for aeroelastic or 6DOF cases since motion and flow solution are coupled
- Use these with some form of animation output: only *solid boundary* output is appropriate for `body_motion_only`; with `grid_motion_only` can look at any boundary, or use sampling to look at interior planes, etc.



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



25

List of Key Input/Output Files

- Beyond basics like `fun3d.nml`, etc.:
 - Set `moving_grid = .true.` in `&global` namelist
- Input
 - `moving_body.input` (else code stops when `moving_grid = T`)
- Output
 - `[project]_subhist.dat`
 - `PositionBody_N.dat` (forced motion / 6-DOF only)
 - `VelocityBody_N.dat` (forced motion / 6-DOF only)
 - `AeroForceMomentBody_N.dat` (forced motion / 6-DOF only)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop
June 20-21, 2015



26



Using Suggar++ With Unstructured Grids

Ralph Noack, Ph.D.
President

Celeritas Simulation Technology, LLC

www.CeleritasSimTech.com

1



Outline

- Brief overview of overset approach
- Why use Overset?
- Introduction to Suggar++ Inputs
- Summary

2



Overview of Overset



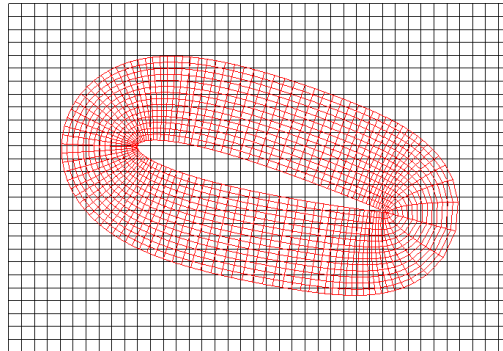
Overset / Chimera Fundamentals

- Set of body fitted grids (structured or unstructured) are constructed around each component of a complex configuration
- Component grids are constructed (mostly) independently from each other
- Overlap each other arbitrarily within each component, and between all components
- Interpolation links solution on component grids



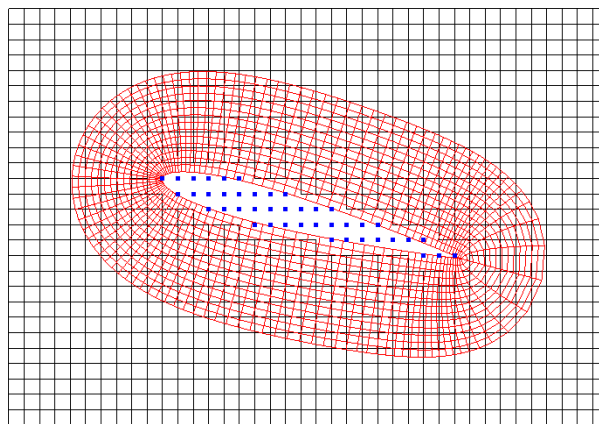
Overset Grid Approach

- Domain is discretized using overlapping component grids
- **Overset composite grid** consists of
 - Composite grid (set of component grids, possibly treated as a single unstructured grid)
 - Domain Connectivity Information (DCI)



Overset Hole Cutting

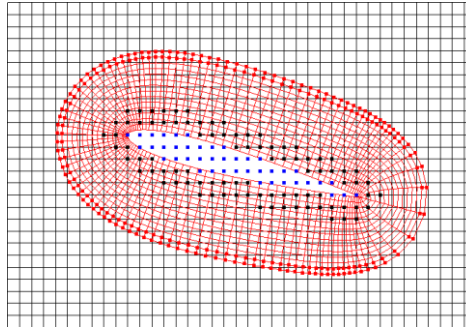
- **Hole cutting** is required to identify points that should be excluded from computations (OUT points)





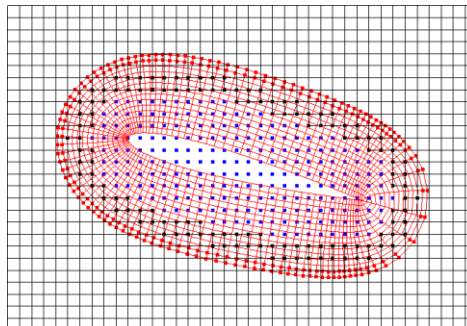
Overset Fringe Boundary Points

- **Intergrid** or **Fringe boundary points** connect the solutions on different components
- **Inner fringe** between hole points and active solution points
- **Outer fringe** at outer/overlap boundary



Overset Overlap Minimization

- Solution quality can be improved by reducing the overlap between grids
- Goal is for fringe (receptor) and interpolation source (donor) to have approximately the same size





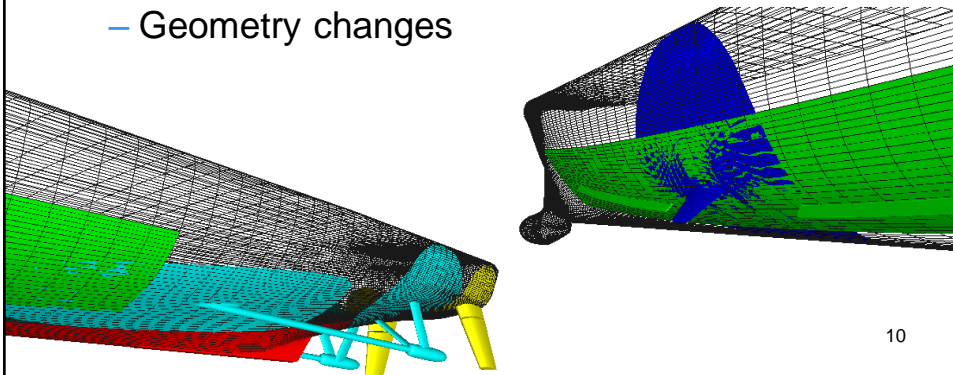
Why use Overset?

9



Simplify Grid Generation

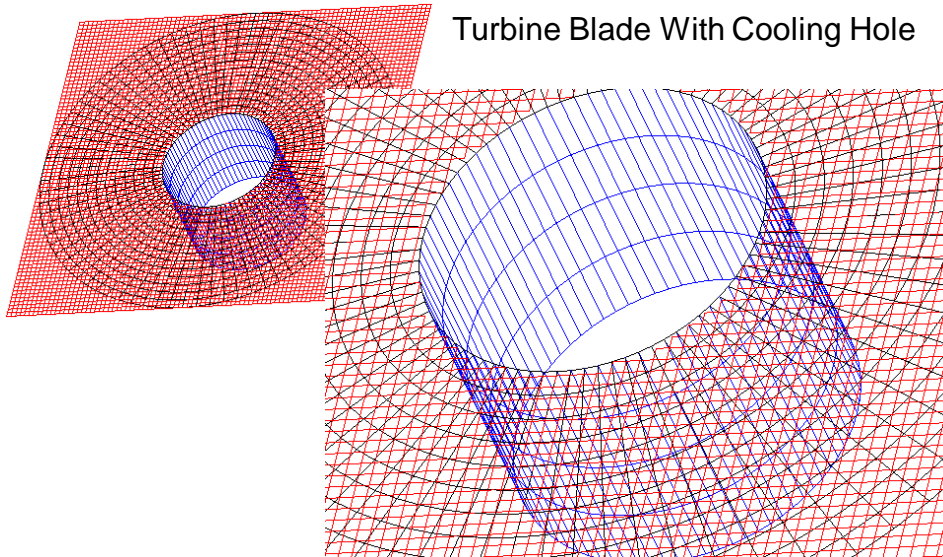
- Flexibility of overlapping grids simplifies grid generation
 - For complex geometries
 - Geometry changes



10

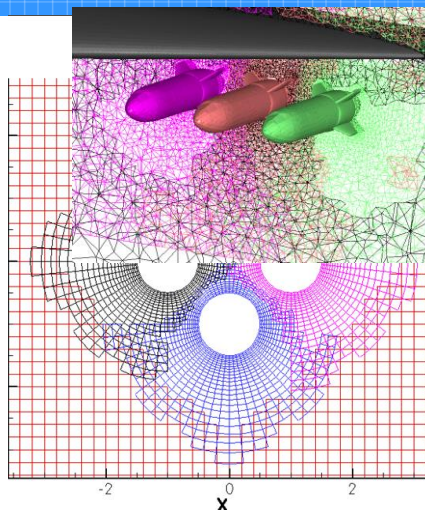


Example of Flexibility Easily Modify Geometry



Example of Flexibility Multiple Copies of Geometry

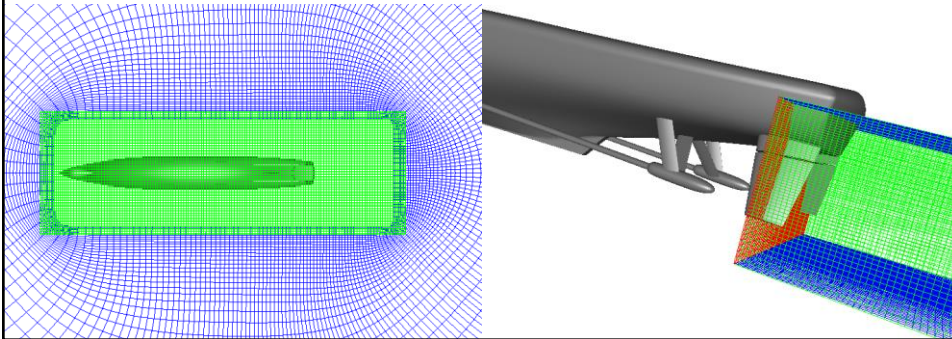
- Multiple identical bodies can be easily gridded by simply copying and translating.
 - Consistent grid in all copies
- Important for evaluating multiple variants quickly.





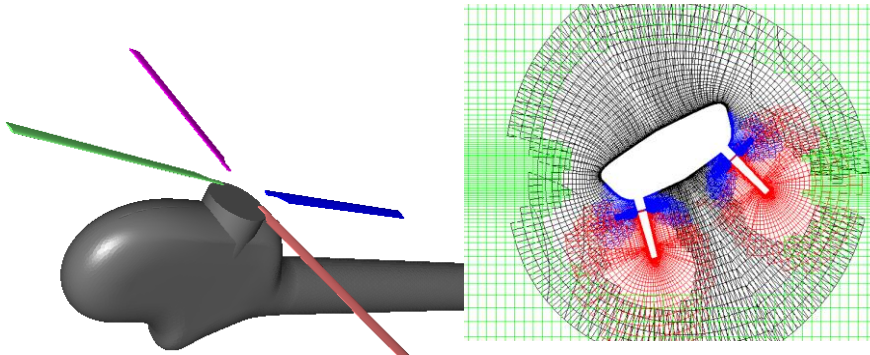
Improving Resolution

- Overset grids simplifies improving resolution in appropriate locations
 - Insert a new grid with desired refinement



Enabling Relative Motion

- Overset grids will move rigidly with each body
- Domain connectivity must be recomputed



14



Overset Grid Generation “Requirements”

15



Overset Grid Generation “Requirements”

- **Need sufficient overlap between grids**
- Better flow solution when
 - Cell size is consistent in overlap region
 - Fringe & donor have similar sizes
 - Do not have large regions of overlap
 - Use overlap minimization procedure to trim excess overlap

16



Suggar++ Inputs

17



XML Tags/Markup Constructs

- An XML tag is enclosed in "< >"
 - `<start>`
 - Must have an associated end tag
 - Same as start tag but with / after <
 - `</start>`
- ```
<name>
 <first>John</first>
 <last>Doe</last>
</name>
```
- Empty elements can have implicit end tag
    - `<name></name>` can be written as `<name/>`



## Hierarchies in XML

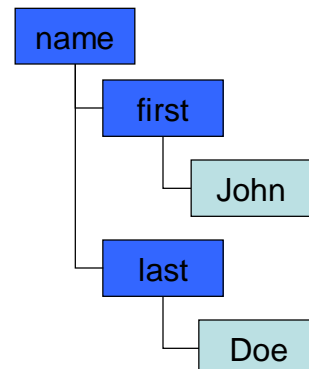
- Each XML tag defines an item or **element**
- Elements can be embedded inside start/end pair of another element
  - Creates a parent/child and sibling/sibling relationship
  - Children define element content
  - Child element must be closed before a parent can be closed
- Only one root element allowed



## Example Hierarchy

- Hierarchy for `<name>` example

```
<name>
 <first>John</first>
 <last>Doe</last>
</name>
```





## XML Elements Can Have Attributes

- **Attributes**
  - are name/value pairs associated with an element
  - are always attached to the start tag
  - must have a value enclosed in quotes (either single or double quotes)
- Place inside of start tag before closing ">"

`<body name="store">`



## Comments in XML

- Comments in XML
  - start with `<!--` and end with `-->`
  - cannot use `--` in the comment string
    - `<!-- cannot embed double dashes -->`
  - cannot be within a tag
    - `<start <!-- this is illegal--> />`



# Suggar++ Input Sections



## Input Has Three Main Sections

- Global parameter
  - Content of <global>
- Body Hierarchy
  - <body>
- Grid/Surface definition
  - <volume\_grid>
    - <boundary\_surface>



## Values Specified by Attributes

- All input values are specified by element attributes
  - `<body name="root">`
  - Data between elements (PCDATA) is ignored
    - Can use as comments, some restricted characters
- Some attributes are required
  - Will abort if not present
- Other attributes are optional

25



## Global Inputs



## <global> Content

- Specify execution control parameters
- Sets default values for parameters that can be set in a grid
- See Suggar++ user's guide for complete list
- Specify root body

27



## <global> Typical Content

```

<global>
 <threads n="5"/>
 <hole_cut method="direct" fill_type="out_cells"/>
 <minimize_overlap set_dsf="element_size"/>
 <!--<cell_centered mark_using_neighbors="Yes"/> -->
 <output>
 <composite_grid style="afIr3" filename="composite_grid.r8.ugrid"
 precision="double" format="unformatted" />
 <domain_connectivity style="unformatted_gen_drt_pairs"
 byte_order="native" filename="output++.dci"/>
 </output>
 <body name="root"> ... </body>
</global>

```

28





## Composite Grid

- Best if flow solver uses composite grid written by Suggar++
  - Exporting from grid generator has possibility of incorrect order of component grids
- Use `<composite_grid/>` to specify format and filename
- Some restrictions on format of input component grids and output composite grid

29

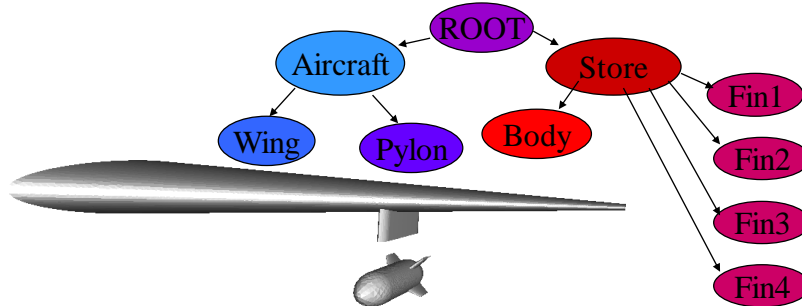


## Body Hierarchy



## Body Hierarchy Controls Hole Cut

- A hierarchical grouping of grids/bodies minimizes user inputs and controls which grids are cut by which surfaces
- **Siblings cut each other**
  - Geometry in one body (including all children) cuts all grids in a sibling body (including all children)



## XML for Wing/Pylon/Store Hierarchy

```
<body name="Root">
```

```
 <body name="Aircraft">
 <body name="Wing"/>
 <body name="Pylon"/>
 </body>
```

```
 <body name="Store">
 <body name="Body"/>
 <body name="Fin1"/>
 <body name="Fin2"/>
 <body name="Fin3"/>
 <body name="Fin4"/>
 </body>
```

```
</body>
```





# Transformations



## Transformations

- Associated with a body
- Hierarchical: Child body transforms are relative to parent
- Order dependent
- Suggar++ has two different types of transformations
  - **Static** transformations
    - Applied to the grid coordinates on input
    - Original coordinates are replaced by transformed coordinates
  - **Dynamic** transformations
    - **Flags the body as moving**
    - Grid coordinates are left in original coordinates
      - Transformations are always from original coordinate system
    - Transformations are used internally during execution
    - Output grids are transformed



## Hierarchical Transformation Example

```
<body name="Aircraft">
 <transform>
 <rotate axis="y" value="20"/>
 </transform>

 <body name="fuselage"> ... </body>

 <body name="store">
 <dynamic>
 <transform>
 <rotate axis="z" value="45"/>
 <translate axis="y" value="-2"/>
 </transform>
 </dynamic>
 </body>
</body>
```

Static  
transformation

Dynamic  
transformation



## Component Grid Input



## SUGGAR++ Current Grid Types

- Structured
  - Curvilinear
  - Analytic
    - Cartesian
    - Cylindrical
    - Spherical
- Unstructured
  - Tetrahedron
  - Mixed element
    - Tet, Hex, Prism, Pyramid
  - Octree-based Cartesian

Can use mix of input grid types if solver and output composite grid supports the mixture of elements



## <volume\_grid> Element

- Parent element is `<body>`
- Associates a grid with a body
  - Actual grid to be used is specified with the filename attribute.
- A body can have more than one `<volume_grid>` child
  - **Cannot have child `<body>` and child grids!**
- Required attribute is name="grid name"

```
<body name="Wing">
 <volume_grid name="wing_grid">
 </volume_grid>
</body>
```



<volume\_grid>  
filename, style attributes

- Grid file is specified with the attributes...
  - *filename*="file"
  - *style*="style"
- Both are required

```
<volume_grid name="wing"
 filename="Grids/wing.g" style="p3d"/>
```



## Boundary Surfaces



## Boundary Surface Creation

- Boundary surfaces are automatically created for **unstructured** surface patches
  - User must specify associated Boundary Conditions
  - Boundary conditions are automatically set for VGRID files
    - Internal mapping between USM3D BCs and Suggar++ BCs
    - Can specify an alternate mapping
- Must be explicitly defined for **structured** grids
  - If not defined surface is created with a boundary condition of “overlap”



## Suggar++ Boundary Conditions

- Suggar++ boundary conditions **do not need to “match”** flow solver boundary conditions
- Some cases where there may be a loose mapping
  - Flow solver “wall” ~ Suggar++ “solid”
  - Flow solver “farfield” ~ Suggar++ “farfield”
  - Geometric connections: axis, Block-to-Block, etc.



## Suggar++ Boundary Conditions

- Many cases where Suggar++ BCs should be different than solver BCs
  - Hole cutting geometry must be closed/“water tight”!!!
    - Surface is not solid geometry but must be used as hole cutting geometry
      - Inlet/Exhaust surface
  - Solver has solid surface that is not needed as cutting surface
    - Tunnel walls but no grids extend past tunnel walls
  - Suggar++ has a limited set of BCs



## Specifying Boundary Conditions for Unstructured Grids

- Boundary conditions can be specified
  - in the input XML file
    - `<boundary_surface find="yes" name="...">`
  - in auxiliary files (Recommended approach)
    - for Vgrid file sets
      - `projectName.suggarbc`
    - for other unstructured grid files
      - `gridFilename.suggar_surface_bc`
      - `gridFilename.suggar_mapbc`
- An auxiliary file can also be used to specify solver BCs in the output composite grid
  - `filename.solver_bc`





## SUGGAR++ Boundary Condition Types

- “**overlap**” An overset or overlap boundary surface.
- “**solid**” A solid boundary and will be used to define the hole cutting geometry.
- “**symmetry**” A symmetry non-overset boundary surface. The grid points on the symmetry boundary will be used to determine the value of the symmetry plane.
- “**axis**” A singular axis where all the grid points in one of the computational coordinates are collapsed to a point.
- “**periodic**” A periodic boundary in the structured grid. Both the min and max boundary surfaces should be specified.
- “**cut**” The surface is a cut boundary in the structured grid. Both the min and max boundary surfaces should be specified.
- “**block-to-block**”, “**block-block**”, “**block2block**” The surface is a block-to-block interface to another grid. Requires additional attributes.
- “**freestream**” or “**farfield**” A freestream non-overset boundary surface
- “**non-overlap**”, “**non\_overlap**”, “**nonoverlap**”, “**non-solid**”, “**non-\***” The surface is an unspecified non-overset boundary.



## Setting **Solver** BCs for Unstructured **Component** Grids

- **Solver** BCs can be set from auxiliary files associated with each component grid
  - Vgrid
    - project.mapbc file
  - Cobalt
    - grid\_filename\_cobalt\_bc
    - basename.cobalt\_bc
      - Where **basename** = grid\_filename with trailing suffix removed
  - Other formats
    - grid\_filename.solver\_bc
    - grid\_filename.suggar\_mapbc



## Solver BCs for Unstructured **Composite** Grid

- Suggar++ will write selected solver boundary condition files for the composite grid
  - Vgrid  
`project.mapbc` file
  - Cobalt  
`composite_grid_filename_cobalt_bc`
  - Other unstructured grid formats  
`composite_grid_filename.suggar_mapbc`



## Overlapping Surface Grids



### Overlapping Surface Grids: Additional Complexities

- Overlapping surfaces will have different discrete representations
- Surfaces in a grid can be associated with different geometry components
  - Grid is in one body but some surfaces define geometry in another body
- Overlapping surfaces require special treatment to eliminate double counting in Force and Moment integration



### Overlapping Surface Grids: Different Discrete Representations

- Surfaces that overlap on geometry with curvature will have different discrete representations
- Difficulties arise when the tangential spacing is “large” relative to the curvature and the normal spacing
- Special procedures are required to properly find appropriate donors



## Surface Assembly

- Grid points are not changed
- Fringe points are shifted during the donor search
  - Requires grid with structure normal to the surface
    - Structured grid or mixed element with hex/prism layers
    - Vgrid tet mesh with layers and poin1 file
- Surface assembly procedure is use to find the shift for each fringe point
  - Enabled with <surface\_assembly/> element
  - Relative to overlapping surface in each donor grid
    - A fringe point will have different shifts/offsets for each donor grid



## Force And Moments On Overlapping Surfaces



## Integrating Force And Moments On Overlapping Surfaces

- Special treatment to eliminate double counting in force and moment integration
  - Panel weights
    - Weight factor between 0 & 1 for each integration surface face/panel
  - Single valued/water tight integration surface
    - Remove overlap, glue remaining portions of original surfaces together using new triangles

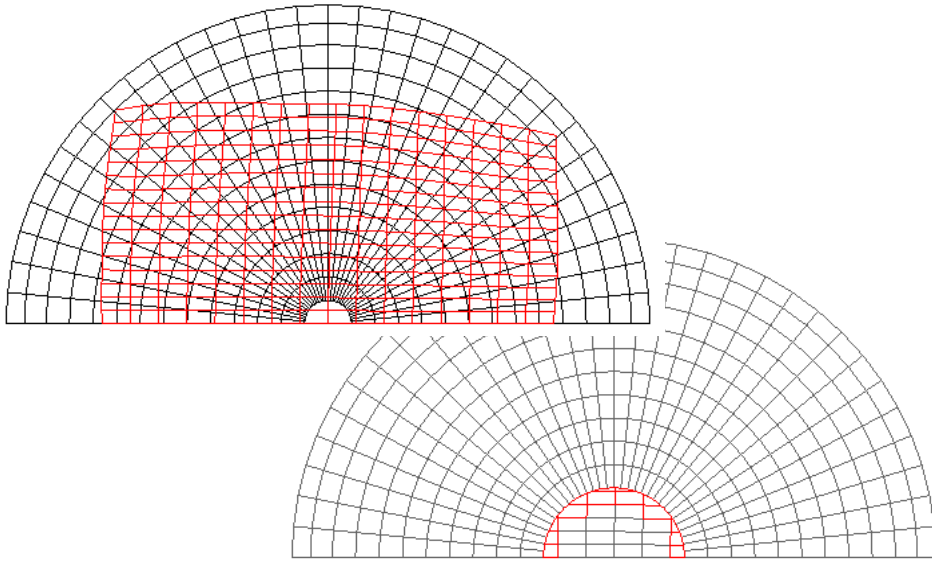


## Suggar++ has integrated USURP capability

- Enabled with <usurp> element
- Output
  - Panel weights
    - Included in DCI file: Can be retrieved via DiRTlib
    - Written to files
  - Can create zipper grid
    - Water tight surface grid with overlap eliminated
    - Not sufficiently robust



## <usurp> Example: Overlapping Surfaces



**EXAMINE SUGGAR++  
RESULTS**



## General Suggestions Check Suggar++ Output

- Look at
  - summary.log
  - Standard error output file
    - *Suggar++ -reopen* will write to out.stderr++
- Visualize the DCI
  - Look at orphans
  - All blanked points
    - May have flood fill leak if entire grid is blanked out
  - Use gviz or pointwise



# SUGGAR++ USER'S GUIDE



## Suggar++ User's Guide

- List of all inputs elements
  - Hyperlinked to parent element
  - Possible child content
- List of attributes
  - Hyperlinked to parent element
- Sections on usage, advance topics, grid formats, etc.

59



## SUMMARY

60





## Summary

- Overset grids are an enabling technology
  - Simplifies grid generation/model changes
  - Enables moving body simulations
- Briefly presented some Suggar++ Inputs
- Briefly discussed overlapping surface grids
  - Need surface assembly if overlapping surfaces are not planar
  - Force & Moment integration needs procedure to eliminate double counting

61



Commercial distribution and support  
for Suggar++ provided by

**Celeritas Simulation Technology, LLC**

**<http://www.CeleritasSimTech.com>**

**Exportable under an EAR-99 license**

62

# **FUN3D v12.7 Training**

## **Session 14: Overset-Grid Simulations**

Bob Biedron



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



1

## **Session Scope**

- What this will cover
  - Static and dynamic simulations in FUN3D using overset meshes and SUGGAR++ /DiRTlib
- What will not be covered
  - SUGGAR++ operation (Covered by Ralph Noack)
- What should you already be familiar with
  - Basic time-accurate and dynamic-mesh solver operation and control



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



2

## Introduction

- Background
  - Many moving-body problems of interest involve large relative motion
    - rotorcraft, store separation are prime examples
  - Deforming meshes allow limited relative motion before mesh degenerates
  - Single rigid mesh allows only one body; no relative motion
  - Use overset grids to overcome these limitations
- Compatibility
  - Requires DiRTlib and SUGGAR++ from Celeritas Simulation Tech.
  - Grid formats: VGRID, AFLR3, FieldView (FV)
- Status
  - Current SUGGAR++ supports unstructured meshes that overlap on solid surfaces, but we have not really exercised this
  - *Overset grids generally limit scalability; not much of an issue for  $O(100)$  cores*



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



3

## Overset - General Info

- Configuring FUN3D for overset
  - Use `--with-dirtlib=/path/to/dirtlib` and `--with-suggar=/path/to/suggar`
  - FUN3D will expect to find the following libraries in those locations:
    - `libdirt.a`, `libdirt_mpic.h.a` and `libp3d.a` (these may be soft links to the actual serial and mpi builds of DiRTlib)
    - `libsuggar.a` and `libsuggar_mpi.a` (may be soft links)
- You will also need a “stand-alone” SUGGAR++ executable in addition to the library files that FUN3D will link to
- Grids
  - A *composite* overset grid is comprised of 2 or more *component* grids - independently generated - but with similar cell sizes in the fringe areas
  - SUGGAR++ assembles the composite grid from the component grids, and determines overset connectivity data for the composite mesh



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



4

## Overset Preprocessing

- Overset simulations starts with an execution of SUGGAR++ to generate a composite grid and initial (t=0) connectivity data
  - When generating component meshes, try to make cell sizes “similar” in the overlap regions - .e.g. by using similar sourcing strengths
  - Create an XML input file for SUGGAR++ (previous session)
    - Use the name of your FUN3D project for the names appearing in `<composite_grid>` and `<domain_connectivity>`
    - Can mix and match component grid types (VGRID, FV, AFLR) and select one of the types for the output composite grid - but note VGRID only supports tetrahedra
  - Run SUGGAR++ and make sure it all works as expected. You should now have a `[project].dci` file; this Domain Connectivity Information file contains all necessary overset data for solver interpolation between the component meshes at t=0



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



5

## Overset Preprocessing (cont)

- For dynamic-grid simulations, there is an additional consideration at the preprocessing stage: either precompute the overset connectivity for *ALL* time steps up front, or do this “on the fly” from within FUN3D
  - Precomputing requires up-front knowledge of the motion - *rules out 6DOF and aeroelastic cases* since the motion depends on the flow solution; *rules out deforming meshes* even if motion known
  - If the case fits these restrictions, from the point of view of flow solver run time, precomputing all connectivity is the most efficient
  - Need to ensure that SUGGAR++ motion will match FUN3D motion
  - Resulting dci files *must* be named `[project]N.dci` for timestep N
- If connectivity is computed at run time (by necessity or for convenience)
  - Computation of overset connectivity is performed on a single processor (the last one)
  - That processor must have enough memory (basically same memory requirements as stand alone SUGGAR++)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015

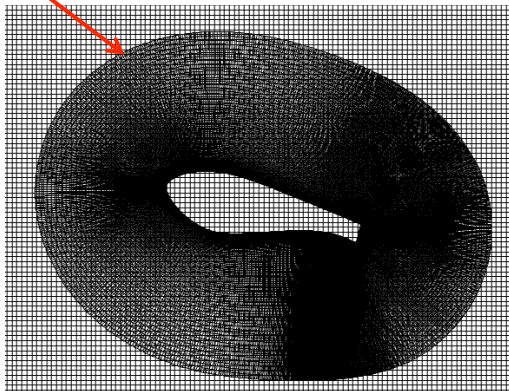


6

## Overset – Boundary Conditions

- FUN3D requires only one specialized overset boundary condition - all other BC's can be applied as needed:
  - In mapbc files, set BC type to -1 for boundaries that are set via interpolation from another mesh

Grid Courtesy Eric Lynch, GA Tech



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



7

## Overset – Boundary Conditions (Cont.)

- SUGGAR++ needs BC info for each component grid
  - Can be set either via the SUGGAR++ input XML file OR an auxiliary file for each component grid
  - Strongly recommend (esp. for dynamic meshes) the XML file approach
    - More cumbersome than auxiliary file, but...
    - If the auxiliary files get separated from the other files, SUGGAR may assume some defaults which can cause problems with hole cutting
    - The exception to setting SUGGAR++ BC info in the XML file is if ALL the component grids are of VGRID type - in that case both SUGGAR++ and FUN3D get BC's from the same VGRID mapbc file and can generally avoid having to explicitly set any BC's for SUGGAR++



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



8

## Overset – Namelist Input

- Control of overset operations - primarily for dynamic grids - set in the `&overset_data` namelist in `fun3d.nml`

```

overset_flag = .true. turn on overset (default: .F.)
dci_on_the_fly = .true. compute connectivity during flow solve (.F.)
reuse_exisiting_dci = .true. if dci file for this step already exists, use it
 instead of computing on the fly (.F.)
dci_period = N dci data repeats every N steps (huge no.)
reset_dci_period = L now repeats every L steps (huge no.)
 ...used for time-step change at restart
dci_freq = M compute dci data every M steps (1)
dci_dir = 'dir' look for or put dci files in this dir (./)
skip_dci_output = .true. don't write dci data after it's computed (.F.)
 ...maybe this data won't be needed again
dci_io = .true. use dedicated proc(s) for fast loading of
 precomputed dci data (.F.) - more later
dci_io_npro = P use P procs for dedicated dci loading

```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



9

## Overset Mesh Simulations – Static (1/2)

- Running FUN3D with static overset meshes:
  - Set `overset_flag = .true.` in the `&overset_data` namelist in `fun3d.nml` (Alt.: use the CLO `--overset`)
  - In screen output, should see something like:
 

```

dirtlib:init_overset Reading DCI data: ./[project].dci
Loading of dci file header took Wall Clock time = 0.002223 seconds
Loading of dci file took Wall Clock time = 0.005657 seconds
Using DiRTlib version 1.49 for overset capability
DiRTlib developed by Ralph Noack, Penn State University Applied Research
Laboratory

```
  - If you request visualization output data for an overset case, “iblack” data will automatically be output to allow blanking of the hole / out points for correct visualization of the solution / grid in Tecplot



<http://fun3d.larc.nasa.gov>

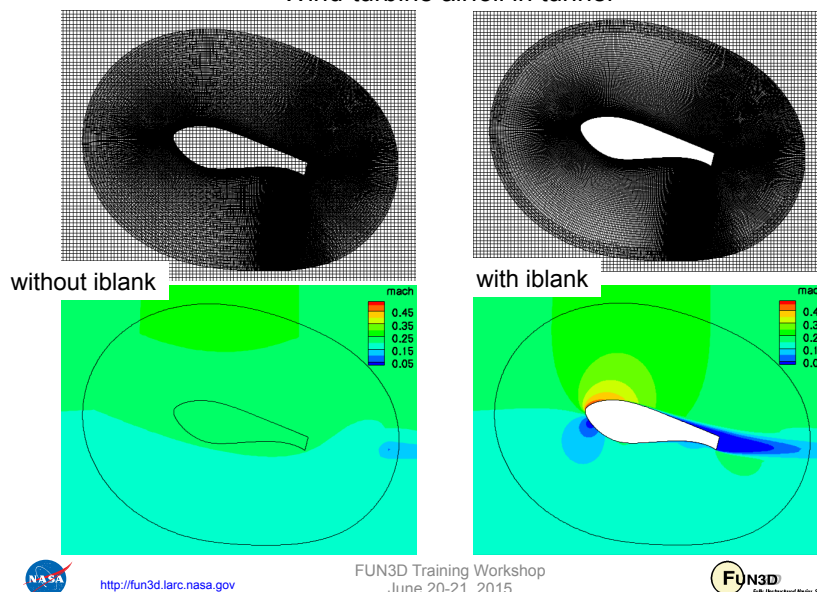
FUN3D Training Workshop  
June 20-21, 2015



10

## Overset Mesh Simulations – Static (2/2)

- Wind-turbine airfoil in tunnel



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



11

## Overset Mesh Simulations – Dynamic (1/5)

- SUGGAR++ setup (more details in “SUGGAR++” session)
  - Starting from a static-grid XML file:
    - Add `<dynamic/>` to `<body>` elements that are to move, e.g.
 

```
<body name="airfoil">
 <dynamic/>
 <volume_grid name="airfoil" style="fvuns"
 filename="airfoil_2p.fvgrid_fmt"/>
</body>
```
    - Note: use a self-terminated `<dynamic/>` so that any `<transform>` elements of `<body>` are applied as static transforms on the component grids when assembling the composite grid
  - Use SUGGAR++ to generate the initial ( $t = 0$ ) composite grid; lets assume you called the XML file `Input.xml_0`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



12

## Overset Mesh Simulations – Dynamic (2/5)

- In the FUN3D `moving_body.input` file
  - Define the bodies and specify motion as usual; boundary numbers correspond to those in the *composite* mesh mapbc file, accounting for any boundary lumping that may be selected at run time
  - Use the component body names from the `Input.xml_0` file
  - Add name of the xml file used to generate the  $t = 0$  composite mesh:
 

```
&composite_overset_mesh
 input_xml_file = 'Input.xml_0'
/
```
- Running FUN3D
  - Set `moving_grid = .true.` in `&global` namelist and `overset_flag=.true.` `dci_on_the_fly = .true.` in `&overset_data` namelist
  - When `dci_on_the_fly = .T.`, FUN3D calls libSUGGAR++ to compute new overset data when the grids are moved; if `.false.` (default), solver will try to read the corresponding dci file from disk



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



13

## Overset Mesh Simulations – Dynamic (3/5)

- Running FUN3D (cont)
  - Note: when `dci_on_the_fly = .true.`, the *component* grids and mapbc files must be available (can be soft linked) in the FUN3D run directory, in *addition* to the  $t = 0$  *composite*-grid and mapbc files
  - When using `--dci_on_the_fly`, specify *one* additional processor for SUGGAR++
    - The *last* processor gets assigned the SUGGAR++ task
    - *This processor must have enough memory for entire overset problem* (same as needed for SUGGAR++ alone)
  - There are a number of other overset-grid CLOs that may be useful for dynamic overset meshes (see “Overset – Namelist Input” slide).



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



14



## Overset Mesh Simulations – Dynamic (4/5)

- Another option, in the `&global_data` namelist in `fun3d.nml`

```
grid_motion_and_dci_only = .true.
```

 (default: `.F.`) step through the mesh motion and compute dci data but don't solve flow eqns.
  - Useful as an easy (not the most efficient) way to precompute dci data while ensuring the motion will match exactly with FUN3D
- Solution data in hole points (governing equations *not* solved at hole pts.)
  - Starts at freestream
  - FUN3D will “fill in” flow data at hole points at each time step by averaging data at surrounding points - eventually replaces freestream
  - Averaging is important for dynamic case so a hole point that suddenly becomes a solve point has something better than freestream as an IC
  - *Best Practice*: use “keep inner fringe” option in SUGGAR++ XML file - retains extra fringe (interpolated) points near hole edges as a buffer of points that become exposed before hole pts. - interp. better than avg.



<http://fun3d.larc.nasa.gov>

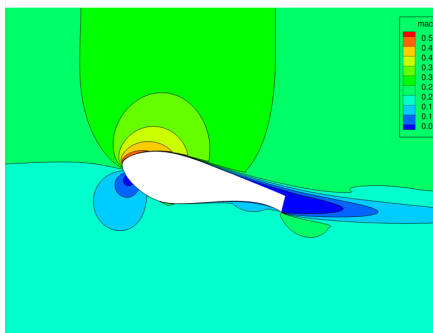
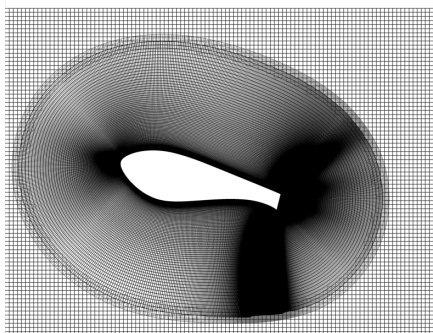
FUN3D Training Workshop  
June 20-21, 2015



15

## Overset Mesh Simulations – Dynamic (5/5)

- Wind-turbine airfoil in tunnel



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



16

## Example – Store Separation (1/5)

- Test case located in: tutorials/flow\_overset\_grids
- Super-coarse grid for a 4-finned store magnetically suspended below a semi-span wing. Could be hooked up to 6DOF library but here we specify the motion for  $t > 0$  as a constant downward velocity
- **run\_tutorial.sh**
  - First runs stand-alone SUGGAR++ executable to generate a dci file for a static-grid / steady-state solution
  - Next runs nodet\_mpi to give a steady-state solution on composite mesh - this will become the starting solution ( $t=0$ ) for the moving-grid / unsteady case
  - Finally runs moving-grid case in which dci data generated “on the fly” for each of 50 time steps



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



17

## Example – Store Separation (2/5)

- Set up SUGGAR++ xml file **wingstore.xml**

```
<global>
 <symmetry_plane axis="Y"/>
 <minimize_overlap keep_inner_fringe="yes"/>
 <output>
 <composite_grid style="unsorted_vgrid_set" filename="wingstore"/>
 <domain_connectivity style="ascii_gen_drt_pairs" filename="wingstore.dci"/>
 </output>
 <body name="wingstore">
 <body name="wing">
 <volume_grid name="wing" style="vgrid_set" filename="wing"/>
 </body>
 <body name="store">
 <dynamic/>
 <volume_grid name="store" style="vgrid_set" filename="store">
 </volume_grid>
 </body>
 </body>
</global>
```

- Add <dynamic/> tag since we will ultimately be doing moving-grid case
- Component grids are VGRID – don't need explicit BCs in the xml file



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



18

## Example – Store Separation (3/5)

- Relevant `fun3d.nml` data (static grid / steady state)

```
&overset_data
 overset_flag = .true.
/
&project
 project_rootname = "wingstore" ! same as <composite_grid> filename
/
 ! we set in wingstore.xml
```

- Relevant `fun3d.nml` data ( moving / unsteady)

```
&overset_data
 overset_flag = .true.
 dci_on_the_fly = .true. ! Must have composite ("wingstore") and
/ ! Component grids ("wing" and "store") in
&global ! in the run directory
 moving_grid = .true.
/
&project
 project_rootname = "wingstore"
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



19

## Example – Store Separation (4/5)

- Relevant `moving_body.input` data ( moving / unsteady))

```
&body_definitions
 n_moving_bodies = 1
 body_name(1) = "store" ! same name used in xml file
 n_defining_bndry(1) = 1
 defining_bndry(1) = 4
 mesh_movement(1) = "rigid"
 motion_driver(1) = "forced"
/
&forced_motion
 translate(1) = 1 ! constant-rate translation
 translation_rate(1) = -0.2 ! Mach 0.2 downward
/
&composite_overset_mesh
 input_xml_file = "wingstore.xml"
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



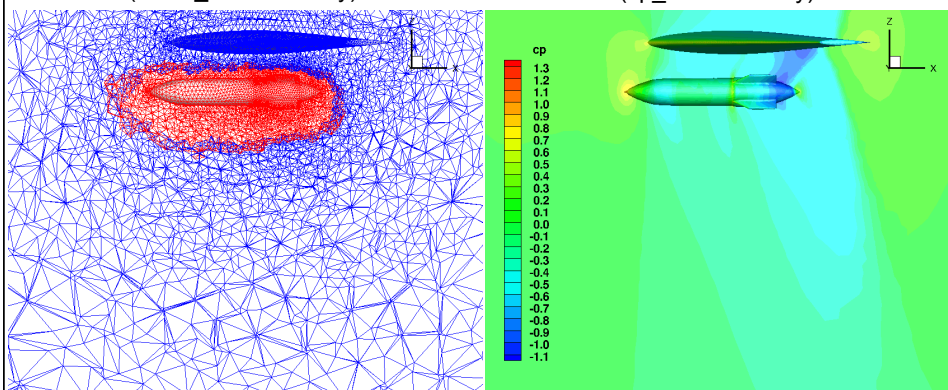
20

## Example – Store Separation (5/5)

Slices Through Store Centerline

Hole Cutting  
(mesh\_animation.lay)

Pressure Coefficient  
(cp\_animation.lay)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



21

## DCI\_IO For Large-Scale Simulations

- Some applications are now run on many-thousand core architectures. SUGGAR++ does not scale well, but for *rigid meshes with prescribed motion*, it is possible to precompute the connectivity data in an “embarrassingly parallel” fashion, avoiding a bottleneck during FUN3D execution
- Normally FUN3D calls DiRTlib routines to load and parse this precomputed dci data. But DiRTlib reads and parses the dci file from every processor, which prohibits scalability beyond ~1k cores
- Instead, use `dci_io = .true.` and use `dci_io_nprocs = P` to assign P processes to read and distribute the dci data - circumvents DiRTlib
  - this is the *only* job for these processors - they operate 1 to P time steps ahead; regular flow-solve ranks work to advance flow in current step
- DCI\_IO utilizes a special file containing a subset of dci data - “dcif” file
  - Convert dci generated by SUGGAR++ to dcif using `utils/dci_to_dcif`
- Linear scaling demonstrated up ~4K cores; P = 1 sufficient for this size



<http://fun3d.larc.nasa.gov>

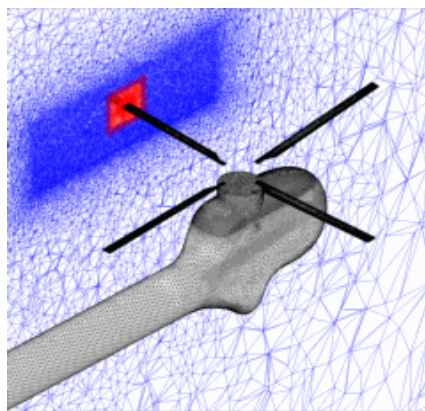
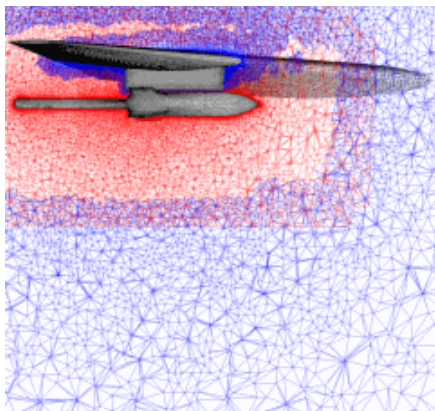
FUN3D Training Workshop  
June 20-21, 2015



22

## Overset Mesh Simulations – Examples

- As always, can use animation to verify; these were done using Tecplot output from FUN3D



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



23

## Troubleshooting

- Orphan count is an indicator (though hardly precise) of problems - either in setup of SUGGAR++ or a poor mesh
  - Both standalone SUGGAR++ and FUN3D (“on the fly”) report orphan counts
    - should have none “due to hole-cut failures”; nonzero count a good indicator of setup issues
    - orphans “due to donor quality” perhaps an indicator of grid quality or setup
  - Visualization often the best tool to remedy
  - Celeritas’ GVIZ or Tecplot output from FUN3D can help sort out oversight connectivity issues



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



24

# FUN3D v12.7 Training

## Session 15: Adjoint-Based Design for Unsteady Flows

Eric Nielsen



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



1

## Learning Goals

- The challenges of unsteady adjoint-based design
- Additional inputs for unsteady design
- Example problem: Maximize L/D for a pitching wing
- Application examples

What we will *not* cover

- Extensive details on setting up the most general problems



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



2

## The Challenges of Unsteady Adjoint-Based Design

### *Sheer Expense*

- The adjoint approach still provides all of the sensitivities at the same cost as analysis, and the 20x estimate still applies for the expense of an optimization
- But every simulation is now an unsteady problem
- Where the steady adjoint solver linearized about a single solution (the steady-state), the unsteady adjoint solver must essentially do this at every physical time step



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



3

## The Challenges of Unsteady Adjoint-Based Design

### *Big Data*

- Since the adjoint must be integrated backwards in time, this implies that we have the forward solution available at every time plane
  - Brute force it: Store the entire forward solution
  - Recompute it: Store the forward solution periodically and recompute intermediate time steps as needed
  - Approximate it: Store the forward solution periodically and interpolate intermediate time planes somehow



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



4

## The Challenges of Unsteady Adjoint-Based Design

### Big Data

*In FUN3D, we store all of the forward data to disk*

- The amount of data adds up fast – consider an example:
  - 50,000,000 grid points and 10,000 physical time steps
  - Using a 1-equation turbulence model (6 unknowns per grid point)
  - Dynamic grids (3 additional unknowns per grid point)
    - $50,000,000 \times 10,000 \times (6+3) \times 8 \text{ bytes} = 36 \text{ Terabytes}$
- So far, this amount of data has not been prohibitively large for our resources, but it is a lot (and we need to go bigger)
  - Will need to tackle this in the long-term
- So far, the challenge has been efficiently getting the data to/from the disk at every single time step



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015

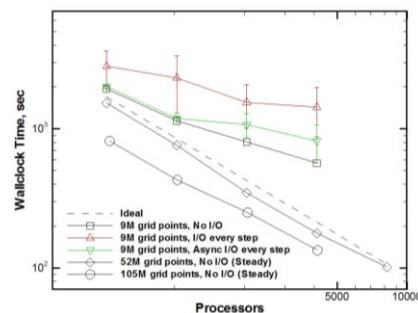


5

## The Challenges of Unsteady Adjoint-Based Design

### Big Data

- Conventional approaches used to write restart files are prohibitively expensive
- System should have a parallel file system
- FUN3D uses parallel, asynchronous, unformatted direct access read/writes from every rank
  - Flow solver is writing the previous time plane while the current time step is computing
  - Adjoint solver is pre-fetching earlier time planes while the current time step is computing
- This strategy performs well for the problems we have run, but is not infinitely scalable



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



6



## The Challenges of Unsteady Adjoint-Based Design

### *Extensive Linearizations*

- If dynamic grids are involved, all of the unsteady metrics and mesh motion/deformations must be differentiated at each time step
- If overset dynamic grids are involved, the relationship between the component grids must also be differentiated at each time step – both motion and interpolants
- If another disciplinary model impacts the CFD model, then that other discipline must also be differentiated, as well as the coupling procedure between the two



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



7

## The Challenges of Unsteady Adjoint-Based Design

### *The Chaos Problem*

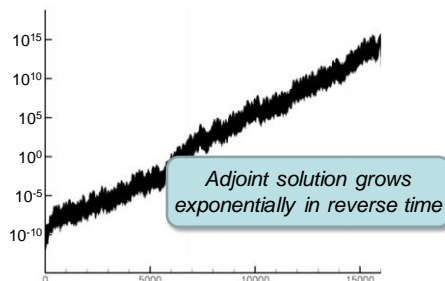
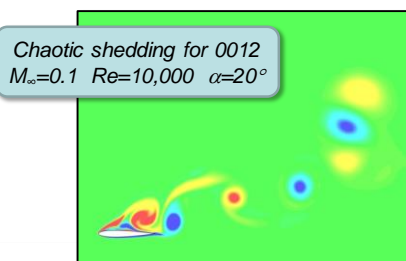
*Wish to compute sensitivities of infinite time averages for chaotic flows (DES, HRLES, LES...)*

- Theory exists that states these sensitivities are well-defined and bounded

#### Why does conventional approach not work?

For chaotic flows:

- The finite time average approaches the infinite time average
- The sensitivity for a finite time average does not approach the sensitivity for the infinite time average



## The Challenges of Unsteady Adjoint-Based Design

### *The Chaos Problem*

- Least-Squares Shadowing (LSS) method proposed by Wang (MIT) and Blonigan (MIT; former LaRC student)
  - Key assumption is ergodicity of the simulation: long time averages are essentially independent of the initial conditions
  - Also assumes existence of a shadowing trajectory
- The LSS formulation involves a linearly-constrained least squares optimization problem which results in a set of optimality equations
- **The LSS adjoint equations are a globally coupled system in space-time**
- To date, work at MIT has focused on solutions of this system for academic dynamical systems containing  $O(1)$  state variables
- Close collaboration between LaRC and MIT is exploring the extension to CFD systems: *enormous* computational challenge for even the smallest of problems



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015

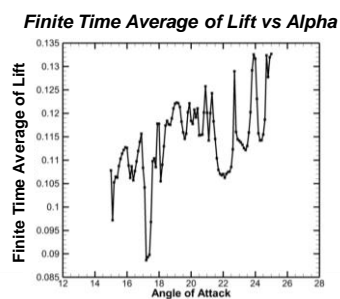
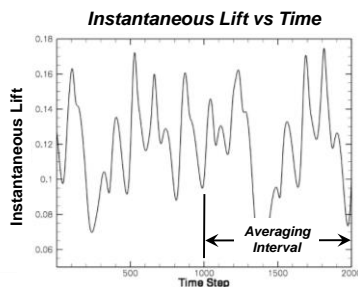
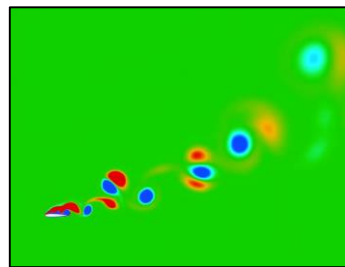


## The Challenges of Unsteady Adjoint-Based Design

### *The Chaos Problem*

Shedding NACA 0012  
 $M_\infty=0.1$   $Re=10,000$   $\alpha=20^\circ$   
 102,940 grid points

- Goal is to compute an AOA sensitivity that would allow us to maximize the time-averaged lift over final 1,000 time steps



## The Challenges of Unsteady Adjoint-Based Design

### *The Chaos Problem*

- FUN3D used to output data for use in LSS solver
  - Nonlinear residual vectors; Jacobians of residual, objective function
  - For this tiny problem, this is 1.1 TB of raw data
- Dimension of the resulting LSS matrix problem:
  - 102,940 grid points x 5 DOFs
  - x 2,000 time planes = 1.03 billion
- Stand-alone LSS solver has been developed where decomposition is performed in time with a single time plane per core
- Global GMRES solver used with a local ILU(0) preconditioner for each time plane



***Just tip of the iceberg – desired simulations are  $10^6$  larger!***  
***Desired matrix dimension =  $10^9 \times 10^6 = 10^{15}$***

## Additional Inputs For Unsteady Design

### *Design Variables*

- All design variables available for steady flows are also available for unsteady flows
- Design variables for a body may now also include FUN3D's rigid motion parameters
- Also have infrastructure for other variables such as boundary condition parameters (e.g., blowing/suction rates), pilot inputs (collective, cyclics) for rotor trimming, etc



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



12

## Additional Inputs For Unsteady Design

### Custom Kinematics

- Design of custom kinematics: users may provide their own routine with a time-dependent  $\mathbf{T}(\mathbf{D})$  matrix governing an individual body's motion
  - Written in complex-variable form, FUN3D will determine its Jacobians automatically

```
!===== USER_SUPPLIED_T =====80
!
! Provides route for user to supply a custom T matrix as a function of time
! and design variables. Complex-valued variables enable automated jacobian
! evaluation.
!
!=====80
subroutine user_supplied_t(ndv,current_time,dvs,t,xcg,ycg,zcg)

 use kinddefs, only : dp

 integer, intent(in) :: ndv

 complex(dp), intent(in) :: current_time
 complex(dp), intent(out) :: xcg, ycg, zcg

 complex(dp), dimension(ndv), intent(in) :: dvs
 complex(dp), dimension(4,4), intent(out) :: t

 continue

end subroutine user_supplied_t
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



13

## Additional Inputs For Unsteady Design

### Objective/Constraint Functions

- The unsteady implementation supports two forms of objective/constraint functions
- The first is based on an integral of the functional form  $f$  introduced for steady flows:

$$f_i = \sum_{n=N_i^1}^{N_i^2} f_i^n \Delta t$$

- The second form is similar, but is based on time-averaged quantities:

$$f_i = \left[ \left( \frac{1}{(N_i^2 - N_i^1 + 1)} \sum_{n=N_i^1}^{N_i^2} C_i^n \right) - C_i^* \right]^{p_i} \Delta t$$



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



14

## Additional Inputs For Unsteady Design

### Objective/Constraint Functions

- The sign of the cost function/constraint input toggles between the two unsteady function forms
  - Positive sign indicates form #1, negative sign indicates form #2
- In addition to the inputs required for steady simulations, the user must now also provide the time interval over which to accumulate the cost function

```
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 cld 0.000000000000000 1.000 20.00000 2.000
```



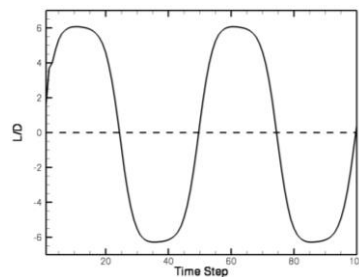
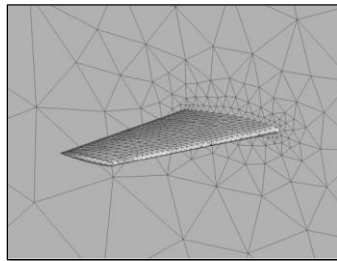
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



15

## Maximize Time-Averaged L/D for a Pitching Wing



- FUN3D's design driver and the optimization packages themselves don't distinguish between steady and unsteady CFD problems – they just see  $f$  and  $\nabla f$
- The problem setup is very similar to steady design cases; will only highlight the differences here



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



16

## Maximize Time-Averaged L/D for a Pitching Wing

### command\_line.options

```
2
2 flow
'--moving_grid'
'--timedep_adj_frozen'
2 adjoint
'--moving_grid'
'--timedep_adj_frozen'
```

- Tell the solvers that it is a moving grid case
- Also specify that we want to do a time-dependent adjoint
  - This kicks in the I/O mechanisms, among other things



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



17

## Maximize Time-Averaged L/D for a Pitching Wing

### moving\_body.input

```
$body_definitions
n_moving_bodies = 1, ! number of bodies in motion
body_name(1) = 'domain', ! name must be in quotes
parent_name(1) = '', ! '' means motion relative to inertial ref frame
n_defining_bndry(1) = -1, ! shortcut to specify all solid surfaces
defining_bndry(1,1) = 1, ! index 1: boundary number 2: body number; use any number for shortcut
motion_driver(1) = 'forced', ! 'forced', '6dof', 'file', 'aeroelastic'
mesh_movement(1) = 'rigid', ! 'rigid', 'deform'
x_mc(1) = 0.25, ! x-coordinate of moment_center
y_mc(1) = 0.0, ! y-coordinate of moment_center
z_mc(1) = 0.0, ! z-coordinate of moment_center
move_mc(1) = 1 ! move mom. cntr with body/grid: 0=no, 1=yes
/
$forced_motion
rotate(1) = 2, ! rotation type: 1=constant rate 2=sinusoidal
rotation_freq(1) = 0.009000, ! reduced rotation frequency
rotation_amplitude(1) = 5.00, ! max rotational displacement
rotation_origin_x(1) = 0.25, ! x-coordinate of rotation origin
rotation_origin_y(1) = 0.0, ! y-coordinate of rotation origin
rotation_origin_z(1) = 0.0, ! z-coordinate of rotation origin
rotation_vector_x(1) = 0.0, ! unit vector x-component along rotation axis
rotation_vector_y(1) = 1.0, ! unit vector y-component along rotation axis
rotation_vector_z(1) = 0.0, ! unit vector z-component along rotation axis
/
```

- Body names must match those specified in rubber.data



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



18

## Maximize Time-Averaged L/D for a Pitching Wing

rubber.data

```
#####
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 0.000000000000000E+01
AOA 0 0.000000000000000E+00 0.000000000000000E+00 0.000000000000000E+01
Number of bodies
1
Rigid motion design variables for 'domain'
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
.
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 Bandalids=2 Sculptor=4)
1
Number of shape variables for 'domain'
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
.
164 0 0.000000000000000E+00 0.000000000000000E+00 0.500000000000000E+01
165 0 0.000000000000000E+00 0.000000000000000E+00 0.500000000000000E+01
166 0 0.000000000000000E+00 0.000000000000000E+00 0.500000000000000E+01
```

- Body names must match those specified in moving\_body.data



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



19

## Maximize Time-Averaged L/D for a Pitching Wing

rubber.data

```
#####
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
51 100
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 cld 0.000000000000000 1.000 20.00000 2.000
```

- Negative sign on function/constraint selection indicates time-averaging form is to be used
- Time step interval for function is also specified

$$f = \left[ \left( \frac{1}{50} \sum_{n=51}^{100} (L/D)^n \right) - 20 \right]^2 \Delta t$$



<http://fun3d.larc.nasa.gov>

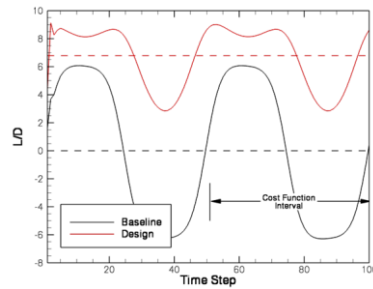
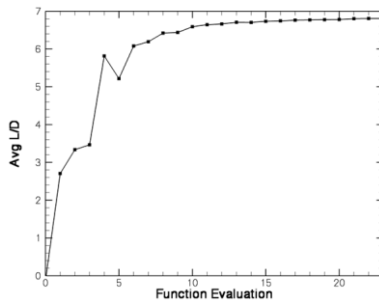
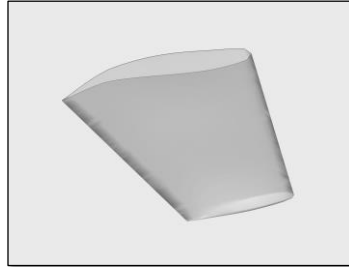
FUN3D Training Workshop  
June 20-21, 2015



20

## Maximize Time-Averaged L/D for a Pitching Wing

- The optimization is executed just as in the steady flow case
- Here, the time-averaged value of L/D has been raised from its nominal baseline value of 0 to an optimized value of 6.8



<http://fun3d.larc.nasa.gov>

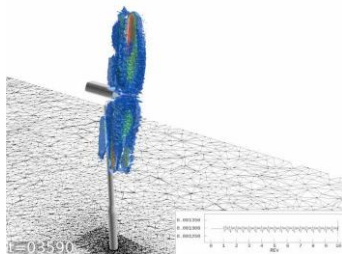
FUN3D Training Workshop  
June 20-21, 2015



21

## Unsteady Design Applications

- This capability is very advanced and can require extensive problem setup for more general, complex applications
- Willing to work closely with someone interested in using it, but fire-hosing you with the intimate details at this point is probably not productive
- Instead, consider some of these prior applications to perhaps spur some ideas on future uses...



**Adjoint Propagating Upstream  
of Wind Turbine**



**Design of Tilt Rotor  
During Pitch-Up**



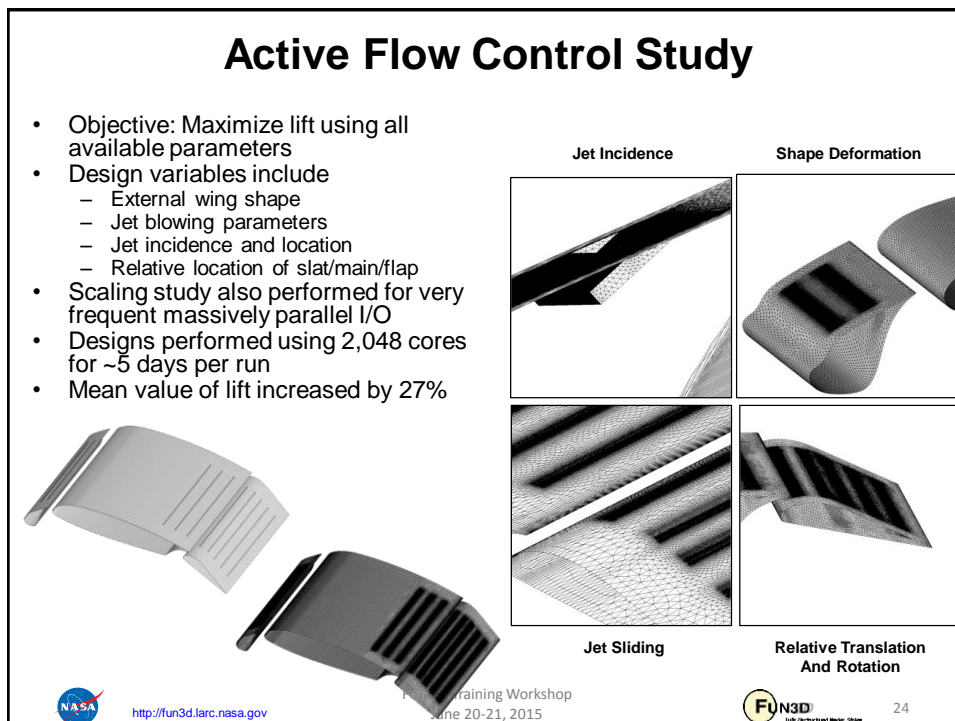
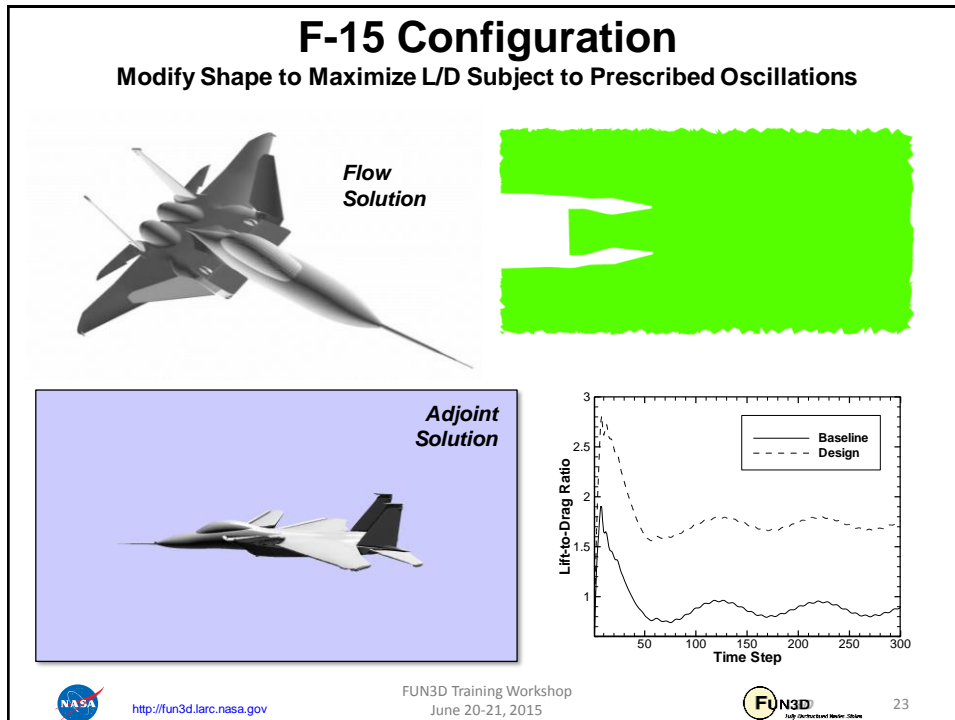
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015

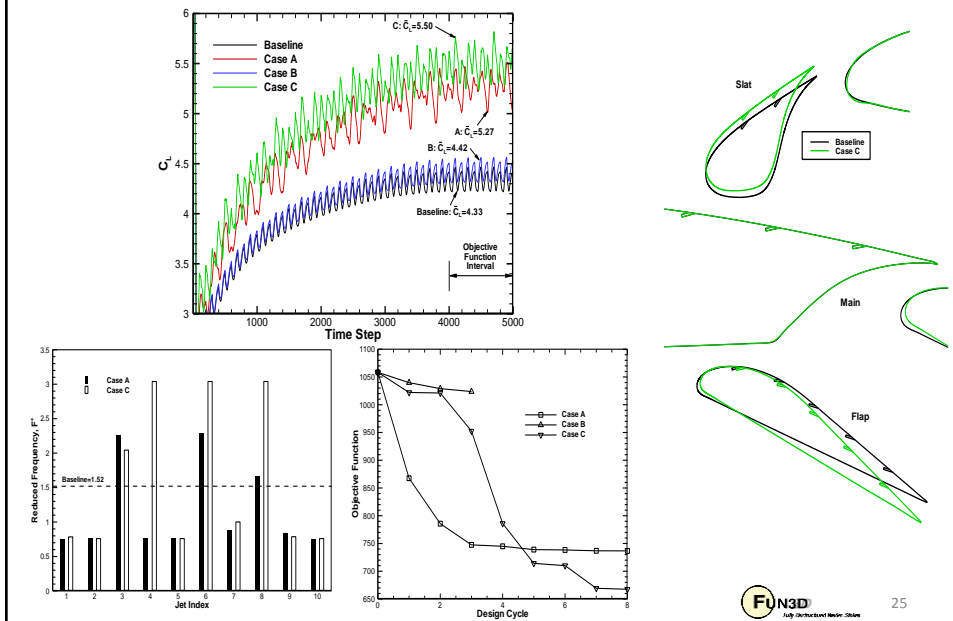


22

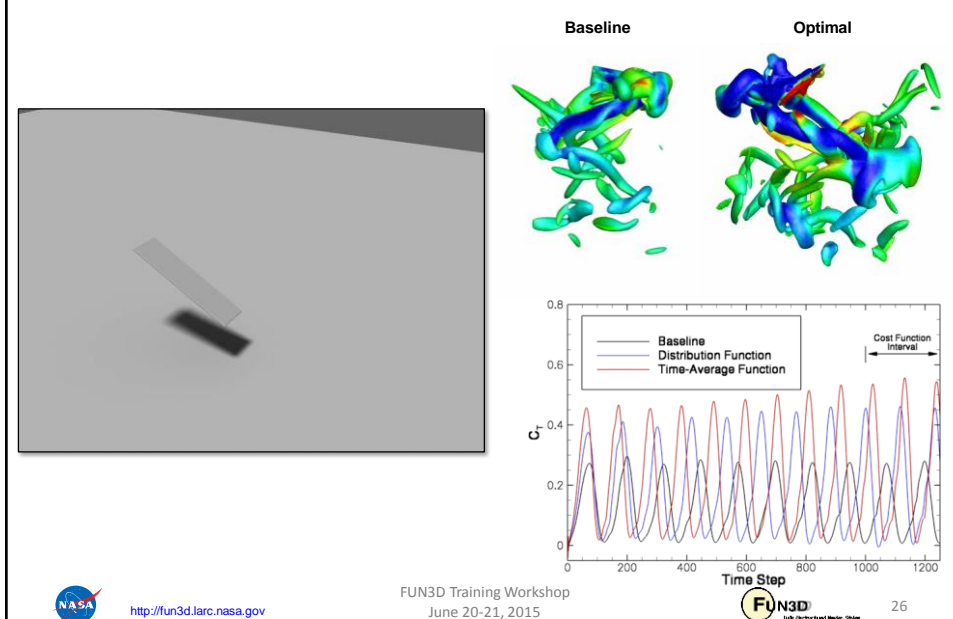




## Active Flow Control Study

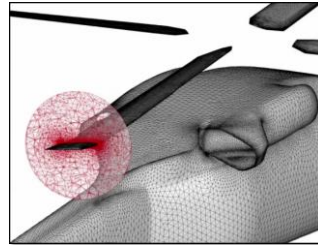
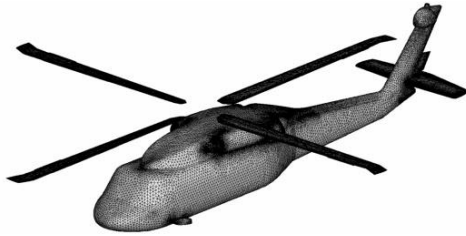


## Flapping Wing Shape & Kinematics



# UH-60 Black Hawk

Maximize Lift Subject to Trimming Constraints



View of Blade Articulation from Blade Reference Frame

$$\theta = \theta_c + \theta_{lc} \cos \psi + \theta_{ls} \sin \psi$$

Blade pitch      Collective      Lateral cyclic      Longitudinal cyclic

- Design variables include blade shape and collective/cyclics
- Three unsteady adjoints computed simultaneously (lift, long/lat moments)



<http://fun3d.larc.nasa.gov>

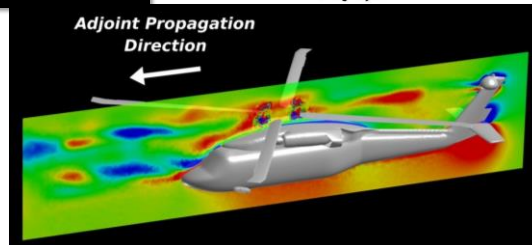
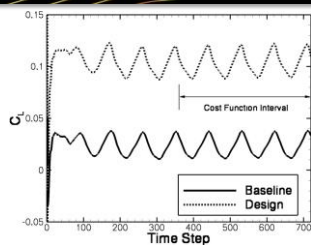
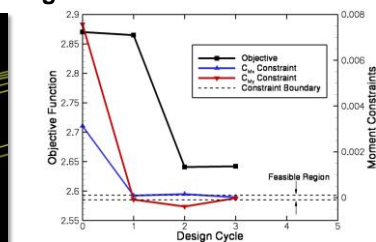
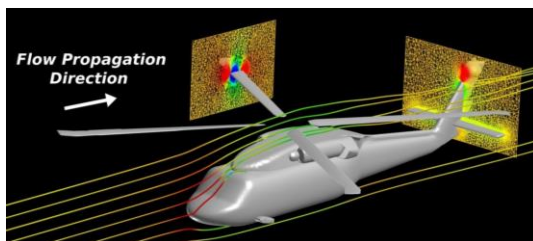
FUN3D Training Workshop  
June 20-21, 2015



27

# UH-60 Black Hawk

Maximize Lift Subject to Trimming Constraints



- Adjoint shows sensitivity of objective function to local disturbances in space and time
- May also be used to perform rigorous error estimation and mesh adaptation
  - Traditional feature-based techniques do not identify such regions



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



28

## List of Key Input/Output Files

### Input

- Same as for steady flows, plus
- `moving_body.input`

### Output

- Same as for steady flows



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



29

## What We Learned

- Challenges involved with adjoint-based unsteady design
- Additional inputs required for unsteady design
- Simple design example for pitching wing
- Previous applications

***Many aspects of this capability are “researchy” and applications of it would benefit from close collaboration***



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



30

# FUN3D v12.7 Training

## Session 16: Aeroelastic Simulations

Bob Biedron



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



1

## Session Scope

- What this will cover
  - The two methods of aeroelastic coupling with FUN3D
    - Static coupling with an external structural solver (linear or nonlinear structures)
    - Dynamic coupling to a self-contained, mode-based, linear structures model
- What will not be covered
  - Projection of mode shapes and forces/displacements to/from CFD and FEM
  - Structural modeling or FEM usage
- What should you already be familiar with
  - Basic steady-state, time-dependent, and dynamic-mesh solver operation and control, especially as pertains to deforming meshes
  - Basic flow visualization



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



2

## Introduction

- Background
  - Aeroelastic problems of interest that can be tackled with FUN3D fall into 2 general categories
    - Static: structural displacement asymptotes to a fixed level; coupling between CFD and CSD can be done infrequently - typically interested in accounting for the structural displacement on (say) cruise performance
    - Dynamic: the change in aero affects the structural deformation to the extent that there is an unsteady coupling between the two; coupling between CFD and CSD must be done frequently - prediction of flutter onset is the classic example
- Compatibility
  - Compatible with compressible flow; mixed elements; 2D/3D
- Status
  - Modal (flutter) analysis fairly routine; static FEM coupling much less so - still evolving; dynamic FEM coupling needs “framework”



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



3

## Static Aeroelastics - Overview

- Basic process (*not a moving\_grid problem - no moving\_body.input*)
  1. Solver starts with an initial grid and solution
  2. Solver reads in a new surface shape and deforms the mesh to fit
  3. Solver performs the requested number of iterations, and outputs aerodynamic loads to a file
  4. Middleware maps aerodynamic loads at CFD grid points onto FEM grid
  5. Structural solver computes new displacements from the airloads
  6. Middleware maps structural displacements onto new surface
  7. Back to step 2; repeat until converged - airloads and displacements
- Historically, Jamshid Samareh of NASA Langley provided middleware (“DDFdrive”) for this loads and deflection transfer; release of this software transitioning to FUN3D team – contact [FUN3D-Support@lists.nasa.gov](mailto:FUN3D-Support@lists.nasa.gov)
- In principle, the above could be applied every time step of a dynamic aeroelastic case; however, file I/O is very inefficient for this



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



4

## Static Aeroelastics - New Surface Shape

- Reading of the updated surface(s) is triggered by the CLO
  - read\_surface\_from\_file**
    - File(s) read once at the start of solver execution (steady-state mode)
    - File root name must be of the form **[project]\_bodyN** (for body N)
    - File extensions: **.dat** or **.ddfb**
      - **[project]\_bodyN.dat** ASCII Tecplot file, “FEPOINT” style
      - **[project]\_bodyN.ddfb** Binary (“stream”) DDFdrive style
      - DDFdrive middleware supports both - **.ddfb** preferred
    - File provides new x,y,z coordinates for each surface point plus an integer that identifies the point in the volume-mesh numbering system
    - Options for this *input* surface file input are specified in the **&massoud\_output** namelist (details later)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



5

## Static Aeroelastics - Aero Loads Output

- Output is triggered by the CLO **--write\_aero\_loads\_to\_file**
  - File(s) written at a user-controlled frequency
  - File root name of the form **[project]\_ddfd drive\_bodyN** (N<sup>th</sup> body)
  - File extensions: **.dat** or **.ddfb**
    - **[project]\_ddfd drive\_bodyN.dat** ASCII Tecplot file, “FEPOINT” style
    - **[project]\_ddfd drive\_bodyN.ddfb** Binary (“stream”) DDFdrive style
    - DDFdrive middleware supports both
  - File provides current  $C_p$ ,  $C_{fx}$ ,  $C_{fy}$ ,  $C_{fz}$  for each surface point plus an identifier that maps the point in the volume mesh
  - Options for this *output* surface file input are specified in the **&massoud\_output** namelist (next)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



6

## Static Aeroelastics - &massoud\_output (1/2)

- The `&massoud_output` namelist serves several closely-related purposes, and the name is not especially well-suited to any of them...
- For static aeroelastics, it is used to
  - Define the aeroelastic body(s) as a collection of boundary surfaces
  - Specify the format of the new surface file and the output aero loads file
  - Specify the frequency of the aero loads output
- Naming convention: `[project]_ddfdrive_bodyN.dat`
- Example:

```
&massoud_output
 aero_loads_file_format = 'stream' (default = 'ascii')
 massoud_file_format = 'stream' (default = 'ascii')
 aero_loads_output_freq = -1 (if +n...output every n steps)
 n_bodies = 1 (default = 0)
 nbndry(1) = 3 (default = 0)
 boundary_list(1) = '1,2,3' (default = '')
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



7

## Static Aeroelastics - &massoud\_output (2/2)

- The `&massoud_output` namelist has additional options
  - rotate, translate and scale the geometry written to the aero loads file
  - multiply the aero coefficients by the dynamic pressure to get forces
  - rotate, translate and scale the geometry read from the new surface file
  - output aero loads on either the deflected or undeflected surfaces
  - See Manual for these infrequently-used options



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



8



## Static Aeroelastics - Towards Automation

- As outlined, the process is rather cumbersome, with multiple separate steady-state runs of the flow solver, the FEM and middleware
  - For our own in-house work we have written a few scripts to orchestrate these steps using DDFdrive
  - Contact [FUN3D-Support@lists.nasa.gov](mailto:FUN3D-Support@lists.nasa.gov) if interested (scripts are not part of standard distribution)
- Longer term, we plan on a “framework” to allow direct access of data between CFD, FEM and middleware to avoid file I/O
  - Helpful for static coupling with an FEM; essential for dynamic coupling
  - Stay tuned for development along these lines



<http://fun3d.larc.nasa.gov>

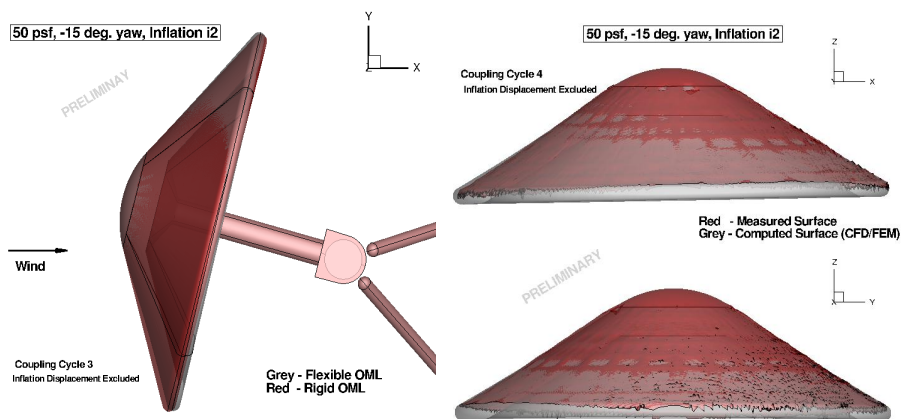
FUN3D Training Workshop  
June 20-21, 2015



9

## Static Aeroelastic Coupling Example

Recent Application: Inflatable Decelerator - Low Speed Test



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



10

## “Bootstrapping” Aeroelastic Problems (1/2)

- All aeroelastic problems, (except highly-specialized rotorcraft problems), utilize either an ASCII Tecplot (.dat) or stream DDFdrive (.ddfb) file to define either a new surface or a set of mode shapes
  - These files need to have the correct surface points for the surface/body in question, plus an integer tag for each point that maps the surface point in the corresponding volume grid.
  - The tag must be preserved throughout any external manipulation of these files (when shape is updated or modes mapped onto surface)
- How does one generate this surface info?
  - Use the CLO `--write_massoud_file` and `&massoud_output` namelist input during an initial run (perhaps when generating a rigid steady-state solution)
  - This will generate a `[project]_massoud_bodyN.(dat or ddfb)` file for input to DDFdrive or as a template for some other middleware.
  - Rename as needed (e.g. `[project]_bodyN` for static AE)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



11

## “Bootstrapping” Aeroelastic Problems (2/2)

- Example

```
&massoud_output
 n_bodies = 2
 nbndry(1) = 3
 boundary_list(1) = '5 7 9'
 nbndry(2) = 2
 boundary_list(2) = '3 4'
/
```

- Also need CLO `--write_massoud_file`



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



12

## Dynamic Aeroelastic Coupling

- For time-accurate aeroelastic modeling, FUN3D currently relies on a modal decomposition approach
  - *Linear* structural dynamics equation (see AIAA 2009-1360) - appropriate for small deflections (e.g. during flutter onset)
  - Deflection assumed a linear combination of eigenmodes (mode shapes)
    - FEM model used a priori to extract eigenmodes / frequencies
    - Deflection represented as linear combination of eigenmodes (mode shapes)
    - Typically only a limited set of the “important” eigenmodes retained
  - A *nonlinear* aerodynamics model is used (FUN3D), so effects of shocks and viscosity can be captured in the flow field
  - Middleware (e.g. DDFdrive) maps eigenmodes onto CFD surface in a one-time preprocessing step; at startup FUN3D reads these
  - Aerodynamics at current time step determine the weight applied to each eigenmode; current shape is weighted sum of eigenmodes



<http://fun3d.larc.nasa.gov>

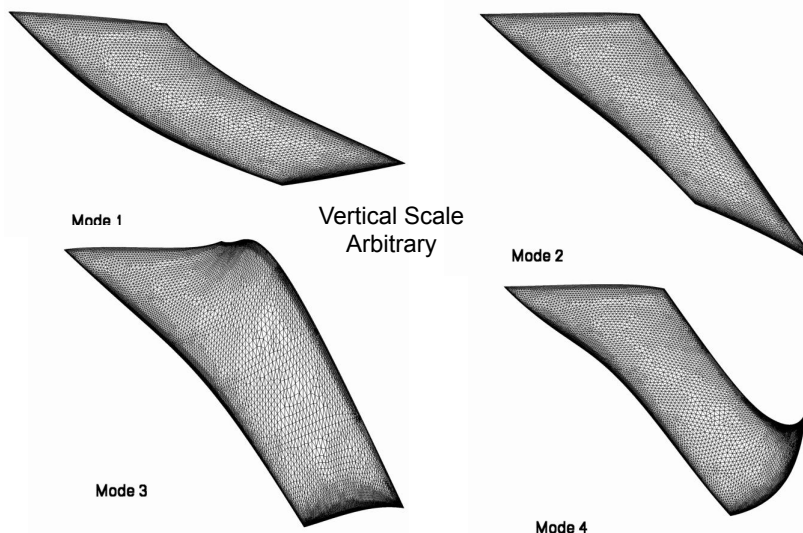
FUN3D Training Workshop  
June 20-21, 2015



13

## Dynamic Aeroelastic Coupling

### First 4 Mode Shapes AGARD 445 Wing



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



14

## Dynamic Aeroelastic Coupling

- Typical flutter assessment process
  1. Run FEM to extract and output the desired modes
  2. Run FUN3D in steady-state mode with `--write-massoud` CLO to generate a steady-state solution and provide a file(s) that will serve as a template for subsequent mode-shape files
  3. Map the FEM modes onto the template (DDFdrive can be used) to generate one surface file per mode
  4. Run FUN3D in moving-grid, time-dependent mode, using modal aeroelastic inputs (upcoming slides) with critical damping ratio  $\sim 1$ 
    - This yields a static aeroelastic deflection, the starting point for flutter assessment
    - Symmetric configuration at zero AoA can skip this step (as in the case in tutorial example covered later)
  5. Run FUN3D in moving-grid, time-dependent mode, using modal aeroelastic inputs with a initial perturbation to “kick” elastic response; does response grow or decay?



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



15

## Dynamic Aeroelastic Coupling

- Steps 4&5 require command-line “option”: `--aeroelastic_internal`
- File nomenclature / format for mode shape input files
  - For every aeroelastic body B, each mode shape M is in a different file: `[project]_bodyB_modeM.dat (.ddfb)`
  - Files are once again either ASCII Tecplot files (`.dat`) or stream DDFdrive files (`.ddfb`), similar to those input for static aeroelastic analysis, only now have modal amplitudes as well:

```
TITLE="wing-445.6 Mode 1"
VARIABLES= "x" "y" "z" "id" "xmd" "ymd" "zmd"
ZONE I= 57286 , J= 101359 , F=FEPOINT
0.109050E+01 -0.650348E+00 -0.294021E-01 17 0.000000E+00 0.000000E+00 0.869050E-01
0.691189E+00 -0.650348E+00 0.000000E+00 18 0.000000E+00 0.000000E+00 0.448300E-01
0.000000E+00 0.000000E+00 0.000000E+00 23 0.000000E+00 0.000000E+00 -0.276958E-02
```

- Can output a “massoud file” from FUN3D (see “Bootstrapping” slide) to use as a template file with x,y,z, and id to which the middleware can add modal amplitudes



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



16

## Tutorial Case: AGARD 445 Wing (1/8)

- Test case located in: tutorials/flow\_modal\_aeroelasticity
  - `run_tutorial.sh` script performs steps 2,3, and 5 of the typical flutter assessment process
    - FEM mode shapes (Step 1) are given in Modes/445.6-mode.dat
    - No need for Step 4: symmetric airfoil at 0 deg. AoA
    - This tutorial case takes several hours to run
- Well-known test case for flutter prediction
  - Tested in NASA Langley Transonic Dynamics Tunnel c. 1960
  - Often run as inviscid (as we do here)
  - Typically run over a range of transonic Mach numbers to see at what dynamic pressure the wing begins to flutter (and with what frequency): here we consider only Mach 0.9 and  $q = 75$  psf



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



17

## Tutorial Case: AGARD 445 Wing (2/8)

- Step 2: Generate template for FUN3D mode-shape files
  - Typical steady-state run, but with CLO `--write_massoud_file`, and `fun3d.nml` namelist input:

```
&massoud_output
 n_bodies = 1
 nbndry(1) = 1 ! Note:family lumping -> 1 boundary
 boundary_list(1) = '3' ! Lumped wing is now boundary no. 3
/
```

- Generates file `wing-445.6_massoud_body1.dat`:

```
title="surface points and l2g id for massoud"
variables="x","y","z","id"
zone t="mdo body 1", i=50827, j=101359, f=fepoint, solutiontime= 0.8000000E+03,
strandid=0
 0.294576800000000E+001 -0.250000000000000E+001 0.212627299999966E-001 1
 0.386499999999999E+001 -0.250000000000000E+001 0.000000000000000E+000 2
 0.254080100000000E+001 -0.209625900000000E+001 0.230393900000010E-001 3
 0.222790400000000E+001 -0.209625900000000E+001 0.000000000000000E+000 4
 ...
```

- FUN3D mode files must preserve these x,y,z and `id` values, and append the x,y,z modal amplitude at the end of each line – `Mode.f` does this



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



18

## Tutorial Case: AGARD 445 Wing (3/8)

- Step 3: Map FEM modal data onto template file generated in Step 2 (one file per mode). In this case we use the custom code Mode.f but could alternatively use a more general tool like DDFdrive

- In this case we end up with 4 files, e.g.

**wing-445.6\_body1\_model1.dat**

```
TITLE="wing-445.6 Mode 1"
VARIABLES= "x" "y" "z" "id" "xmd" "ymd" "zmd"
ZONE I= 50827 , J= 101359 , F=FEPOINT
0.294577E+01 -0.250000E+01 0.212627E-01 1 0.000000E+00 0.000000E+00 0.182049E+01
0.386500E+01 -0.250000E+01 0.000000E+00 2 0.000000E+00 0.000000E+00 0.239954E+01
0.254080E+01 -0.209626E+01 0.230394E-01 3 0.000000E+00 0.000000E+00 0.131437E+01
0.222790E+01 -0.209626E+01 0.000000E+00 4 0.000000E+00 0.000000E+00 0.114554E+01
...
```

- Mode.f has lost a few digits of precision relative to the template, but otherwise preserves the x,y,z and id values; xmd, ymd, and zmd are the mode amplitudes at each point



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



19

## Tutorial Case: AGARD 445 Wing (4/8)

- Step 5: moving\_body.input file:

```
&body_definitions
 n_moving_bodies = 1 ! define bodies as collection of surfaces
 body_name(1) = 'airfoil' ! some name
 n_defining_bndry(1) = -1 ! use all solid surfaces
 motion_driver(1) = 'aeroelastic'
 mesh_movement(1) = 'deform'
/
&aeroelastic_modal_data ! below, b = body #, m = mode number
 plot_modes = .true. ! can tecplot to verify mode shapes read correctly
 nmode(1) = 4 ! 4 modes for this body
 uinf(1) = 973.4 ! free stream velocity (ft/s)
 grefl(1) = 1.00 ! scale factor between CFD and FEM models
 qinf(1) = 75.0 ! free stream dynamic pressure, psf
 freq(1,1) = 60.3135016 ! mode frequency (rad/s)
 freq(2,1) = 239.7975647
 freq(3,1) = 303.7804433
 freq(4,1) = 575.1924565
 gmass(1:4,1) = 4*0.08333 ! generalized mass (nondim)
 gvel0(1:4,1) = 0.1 ! nonzero initial velocity to kick off dynamic
 ! response; set = 0 on restart - don't kick
 ! me twice
/
```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



20

## Tutorial Case: AGARD 445 Wing (5/8)

- Step 5 (cont) Setting the FUN3D timestep
  - From experiment, the flutter frequency at Mach 0.9 is  $\omega^* \sim 120$  rad/sec, so we'll assume we need to resolve at least up to this frequency
  - From nondimensionalization slides, have
    - $t_{chr} = (1/f^*) a_{inf}^* (L_{ref}/L_{ref}^*) = (2\pi/\omega^*) a_{inf}^* (L_{ref}/L_{ref}^*)$
    - $\Delta t = t_{chr}/N$
  - Take 200 steps to resolve this frequency; from previous slide have
    - $U_{inf}^* = 973$  ft/sec so at  $M=0.9$ ,  $a_{inf}^* = 1081$  ft/sec
    - The grid is in ft so  $L_{ref}/L_{ref}^* = 1$
    - $\Delta t = (6.28/120) 1081 (1) / 200 = 0.283$  (tutorial uses 0.3)
    - In practice, would need to do a time step refinement to verify this time step is adequate (at this time step, mode 4 resolved with only ~42 steps/period)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



21

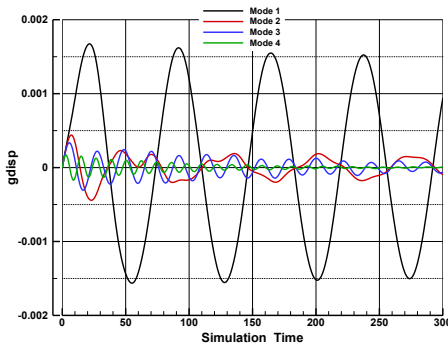
## Tutorial Case: AGARD 445 Wing (6/8)

- Output of generalized force, displacement and velocity into files, e.g. **aehist\_bodyN\_modeM.dat** (ASCII Tecplot)

```
qinf = 7.50000E+01 uinf = 9.73400E+02 Mach = 9.00000E-01
variables = "Simulation_Time", "gdisp", "gvel", "gforce"
zone t = "modal history for airfoil, mode 1"
```

0.00000000E+00	0.00000000E+00	1.00000000E-01	0.00000000E+00
3.00000000E-01	2.77176987E-05	9.98502122E-02	-8.15943032E-02
6.00000000E-01	5.53732953E-05	9.95522312E-02	-7.22530082E-02

Typical plot to assess dynamic response to disturbance (recall initial 0.1 perturbation in gvel)  
At this q, response damps very slowly



<http://fun3d.larc.nasa.gov>

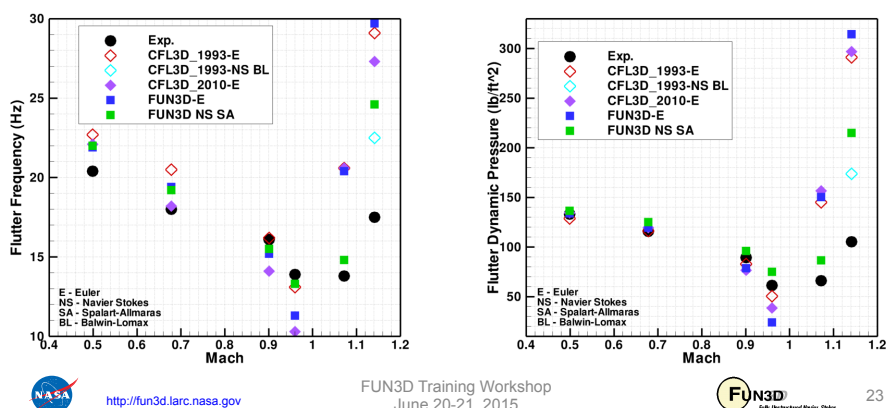
FUN3D Training Workshop  
June 20-21, 2015



22

## Tutorial Case: AGARD 445 Wing (7/8)

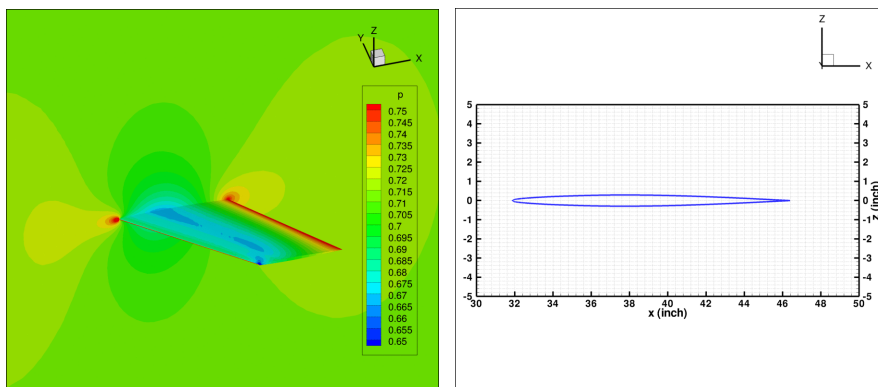
- The dynamic pressure ( $q = 75 \text{ psf}$ ) in the tutorial does not lead to flutter at  $M = 0.9$  – so we would need to increment  $q$  and repeat until we found a response that grows with time ( $M = 78.6 \text{ psf}$ ) – then repeat over the Mach range
- Pawel Chwalowski, Aeroelasticity Branch, NASA Langley has carried out this exercise and provided these plots (not part of tutorial):



## Tutorial Case: AGARD 445 Wing (8/8)

Results Courtesy Pawel Chwalowski, Aeroelasticity Branch, NASA Langley  
(These animations not generated as part of the tutorial)

Inviscid Flow Mach=0.9, Flutter condition,  $Q = 78.6 \text{ psf}$





## Additional Considerations

- Be especially careful with dimensions and coordinate systems since at one point or another exchange must be done between CFD and FEM - need to ensure consistency!
- Note that frequencies increase in the higher modes; choose time steps accordingly



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



25

## List of Key Input/Output Files

- Beyond basics like `fun3d.nml`, `[project]_hist.tec`, etc.:
- Input
  - `moving_body.input`
  - `[project]_body1.dat` (`.ddfb`) (external FEM / static AE)
  - `[project]_bodyB_modeM.dat` (`.ddfb`) (modal structures)
- Output
  - `aehist_bodyB_modeM.dat` (modal structures only)
  - `[project]_ddfdribe_bndryN.dat` (with CLO)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



26

# FUN3D v12.7 Training

## Session 17: Rotorcraft Simulations

Bob Biedron



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



1

## Session Scope

- What this will cover
  - Overview of actuator-disc models for rotorcraft
  - Overview of setup for “first principles” articulated-blade rotorcraft simulations using overset grids
    - Rigid Blades
    - Elastic Blades / Loose Coupling to Rotorcraft Comprehensive Codes
- What will not be covered
  - Rotorcraft Comprehensive Code set up and operation
  - All the many critical setup details for the “first principles” approach
- What should you already know
  - Basic time-accurate and dynamic-mesh solver operation and control
  - Rudimentary rotorcraft aeromechanics (collective, cyclic...)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



2

## Introduction

- Background
  - FUN3D can model a rotor with varying levels of fidelity/complexity
    - As an actuator disk – low-fidelity representation of the rotor - when only the overall rotor influence on the configuration is needed
    - As rotating, articulated-blade system (cyclic pitch, flap, lead-lag), with or without aeroelastic effects - if detailed rotor airloads are needed
  - Trim and aeroelastic effects require coupling with a rotorcraft “comprehensive” code
  - As a steady-state problem for rigid, isolated, fixed-pitch blades in a rotating noninertial frame
- Compatibility
  - Coupled to the CAMRAD II and RCAS comprehensive codes
- Status
  - Far less experience / testing with RCAS than with CAMRAD II
  - Near future: hooks to US Army’s HELIOS rotorcraft framework



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



3

## Time-Averaged Actuator-Disk Simulations (1/3)

- Actuator disk method utilizes momentum and energy equation source terms to represent the influence of the disk
  - Original implementation by Dave O’Brien (GIT Ph.D. Thesis)
  - HI-ARMS implementation (SMEMRD) by Dave O’Brien ARMDEC adds trim and ability to use C81 airfoil tables (*Not covered*)
- Simplifies grid generation – actuator disk is automatically embedded in computational grid (refinement in the vicinity of actuator surface improves accuracy)
- Any number of actuator disks can be modeled
- Requires the `--rotor` command line option (`--hiarms_rotor` for SMEMRD). See `rotor.input` section in user manual for full details



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



4

## Time-Averaged Actuator-Disk Simulations (2/3)

- Different disk loading models available
  - **RotorType** = 1 actuator disk
    - **LoadType** = 1 constant (specified thrust coefficient  $C_T$ )
    - **LoadType** = 2 linearly increasing to blade tip (specified  $C_T$ )
    - **LoadType** = 3 blade element based (computed  $C_T$ )
    - **LoadType** = 4 not recommended, user specified sources
    - **LoadType** = 5  $C_T$  and  $C_Q$  radial distributions provided in a file
    - **LoadType** = 6 Goldstein distribution with optional swirl (specified  $C_T$  and  $C_Q$ )
  - **RotorType** = 2 actuator blades (time-accurate) **Not Functional**



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



5

## Time-Averaged Actuator-Disk Simulations (3/3)

- Actuator disk implementation compatible with the standard steady-state flow solver process (compressible and incompressible)
  - Standard grid formats for the volume grids
  - Standard solver input deck (**fun3d.nml**)
  - Standard output is available (**project.forces**, **project\_hist.tec**, **project\_tec\_boundary.plt**)
  - Want similar solution convergence as a standard steady-state case
- Standard actuator disk model is activated in the command line by **--rotor**
  - Rotor input deck file (**rotor.input**) is required in the local directory
  - **rotor.input** contains disk geometry and loading specifications
  - The disk geometry and loading are output in plot3d format in files **source\_grid\_iteration#.p3d** and **source\_data\_iteration#.p3d**



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



6

## rotor.input File

- Constant/linear loading needs only a subset of the data in the file data (manual defines variables)

```

Rotors Uinf/Uref Write Soln Force Ref Moment Ref ! Below we set Uref = Uinf
1 1.000 1500 0.001117 0.001297 ! Adv Ratio = Uinf/Utip
=== Main Rotor =====
Rotor Type Load Type # Radial # Normal Tip Weight
1 2 50 180 0.0
X0_rotor Y0_rotor Z0_rotor phi1 phi2 phi3
0.696 0.0 0.322 0.00 -0.0 0.00
Utip/Uref ThrustCoff TorqueCoff psi0 PitchHing/R DirRot
19.61 0.0064 0.00 0.0 0.0 0
Blades TipRadius RootRadius BladeChord FlapHinge/R LagHinge/R
4 0.861 0.207 0.066 0.051 0.051
LiftSlope alpha, L=0 cd0 cd1 cd2
0.0 0.0 0.002 0.00 0.00
CL_max CL_min CD_max CD_min Swirl
0.00 0.00 0.00 0.00 0
Theta0 ThetaTwist Thetals Thetalc Pitch-Flap
0.0 0.0 0.0 0.0 0.00
FlapHar Beta0 Betals Betalc
0 0.0 0.0 0.0
Beta2s Beta2c Beta3s Beta3c
0.0 0.0 0.0 0.0
LagHar Delta0 Delta1s Delta1c
0 0.0 0.0 0.0
Delta2s Delta2c Delta3s Delta3c
0.0 0.0 0.0 0.0

```

Key:  
 Required for constant/linear actuator disk  
 Add'l data for blade element or "first  
 principles" simulations  
 (all items must have a value, even if  
 unused)

- Vref=Vtip a bad choice for incompressible – use rotor induced velocity



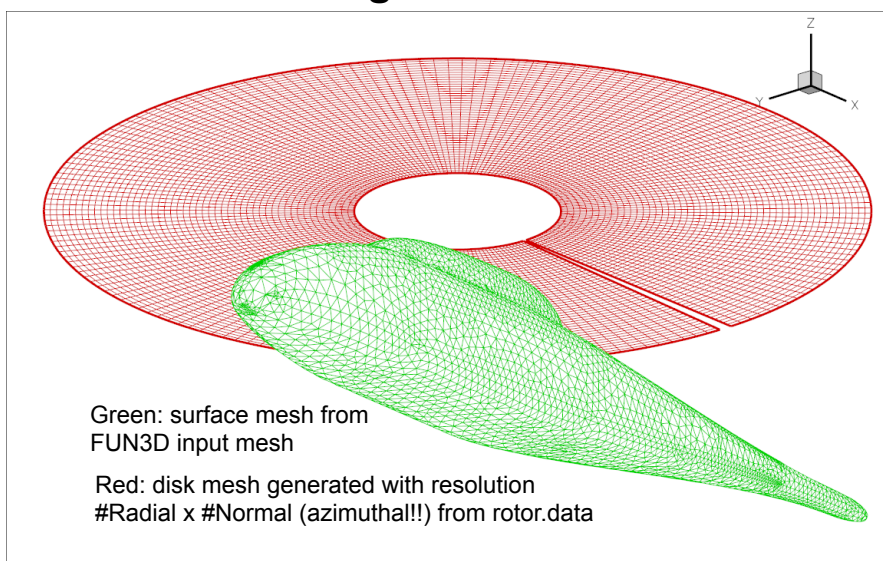
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
 June 20-21, 2015



7

## Robin Fuselage with Actuator Disk



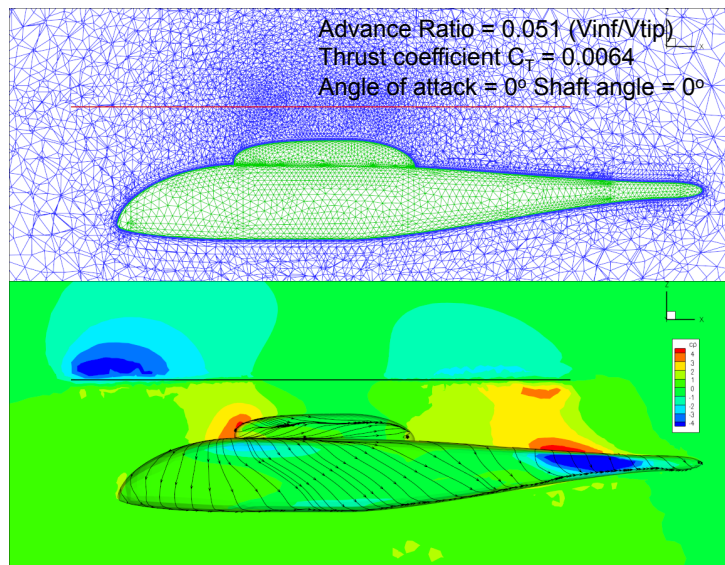
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
 June 20-21, 2015



8

## Incompressible Robin/Actuator Disk



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
 June 20-21, 2015



9

## Articulated-Blade Simulations

- “First Principles” – rotor flow is computed, not modeled
  - Requires moving, overset grids; blades may be rigid or elastic
- Elastic-blade cases (or *trimmed* rigid-blade cases) must be coupled to a rotorcraft Computational Structural Dynamics (CSD, aka comprehensive) code such as CAMRAD or RCAS
  - The CSD code provides trim solution in addition to blade deformations
  - The interface to the CSD code is through standard OVERFLOW `rotor_N.onerev.txt` and `motion.txt` type files
  - Interface codes for CAMRAD are maintained and distributed by Doug Boyd, NASA Langley (contact [d.d.boyd@nasa.gov](mailto:d.d.boyd@nasa.gov))
  - RCAS coupling does not require any interface codes (RCAS API)
  - FUN3D has several postprocessing utility codes tailored to CAMRAD
- Coupled simulations are about as complicated as it gets with the basic FUN3D flow solver
  - There are *many* small details that must be done correctly; we don’t have time to cover them all here



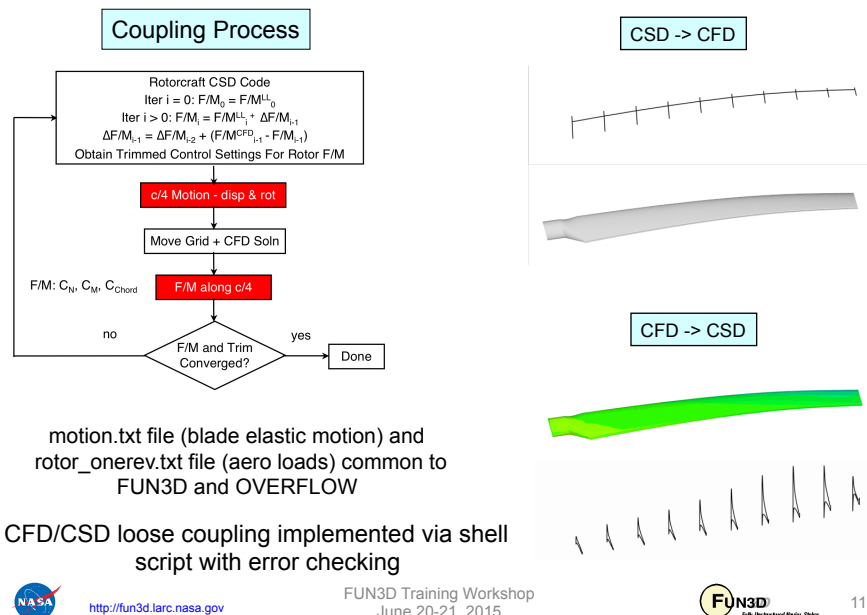
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
 June 20-21, 2015



10

## CFD/CSD – Loose (Periodic) Coupling



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



11

## Rotor-Specific Nondimensional Input (1/2)

- Typically define the flow reference state for rotors based on the tip speed; thus in **rotor.input**, set  $U_{tip}/U_{ref} = 1.0$  (data line 4)
- This way,  $U_{inf}/U_{ref}$  (data line 1) is equivalent to  $U_{inf}/U_{tip}$ , which is the Advance Ratio, and is usually specified or easily obtained
- Since the reference state corresponds to the tip, the **mach\_number** in the **fun3d.nml** file should be the tip Mach number, and the **reynolds\_number** should be the tip Reynolds number
- Nondimensional rotation rate: not input directly, but it is output to the screen; you might want to explicitly calculate it up front as a later check:

$$\Omega^* = U_{tip}^* / R^* \text{ (rad/s, } R^* \text{ the rotor radius)}$$

$$\text{and recall } \Omega = \Omega^* (L_{ref}^* / L_{ref}) / a_{ref}^* \text{ (compressible)}$$

$$\text{so with } a_{ref}^* = U_{ref}^* / M_{ref}^* \text{ and taking } L_{ref}^* = R^*$$

$$\Omega = M_{ref}^* (U_{tip}^* / U_{ref}^*) / R^* \text{ (compressible)}$$

$$\Omega = U_{tip}^* / U_{ref}^* / R^* \text{ (incompressible)}$$



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



12

## Rotor-Specific Nondimensional Input (2/2)

- Nondimensional time step:

$$\text{time for one rev: } T^* = 2\pi / \Omega^* = 2\pi R^* / U_{tip}^* \text{ (s)}$$

$$\text{and recall } t = t^* a_{ref}^* (L_{ref}^* / L_{ref}^*) \text{ (compressible)}$$

so with  $L_{ref}^* = R^*$  we have

$$T = a_{ref}^* (R / R^*) 2\pi R^* / U_{tip}^* = 2\pi R / (M_{ref} U_{tip}^* / U_{ref}^*) \text{ (nondim time / rev)}$$

For N steps per rotor revolution:

$$\Delta t = 2\pi R / (N M_{ref} U_{tip}^* / U_{ref}^*) \text{ (compressible)}$$

$$\Delta t = 2\pi R / (N U_{tip}^* / U_{ref}^*) \text{ (incompressible)}$$

- Note: the azimuthal change per time step is output to the screen in the **Rotor info** section. Make sure this is consistent, to a high degree of precision (say at least 4 digits), with your choice of N steps per rev – you want the blade to end up very close to 360 deg. after multiple revs!
- Formulas above are general, but recall we usually have ref = tip, at least for compressible flow



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



13

## dci\_gen Preprocessor (1/8)

- A rudimentary code to simplify rotorcraft setup (/utils/Rotocraft/dci\_gen)
  - Uses libSUGGAR++ routines
  - Takes a single blade grid and a single fuselage / background grid (extending to far field) and assembles them into an N-bladed rotorcraft
  - Requires **rotor.input** file
  - Creates the SUGGAR++ XML file (**Input.xml\_0**) needed by FUN3D
  - Generates, using libSUGGAR++ calls, the initial (t = 0) dci file and composite grid needed by FUN3D
  - Generates the composite-grid “mapbc” files needed by FUN3D
  - Component grids *must* be oriented as shown on following slide
    - Blade must have any “as-built” twist incorporated
    - If grids do not initially meet the orientation criteria, can use SUGGAR++ to rotate them *before* using **dci\_gen**



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015

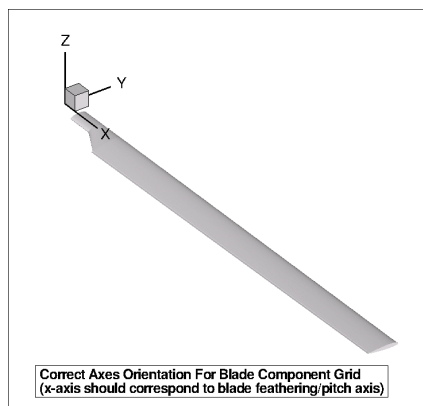
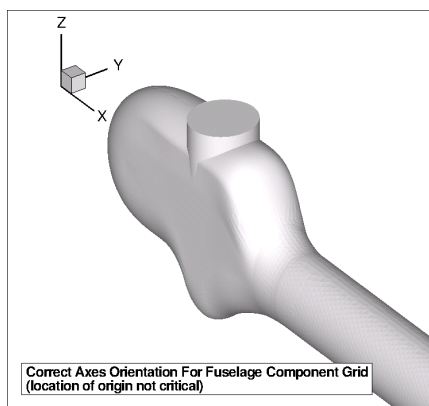


14



## dcgen Preprocessor (2/8)

### HART II Component Grids



<http://fun3d.larc.nasa.gov>

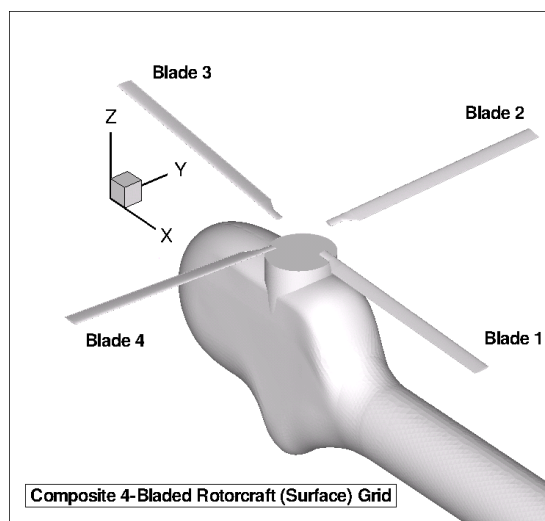
FUN3D Training Workshop  
June 20-21, 2015



15

## dcgen Preprocessor (3/8)

### HART II Composite Grid



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



16

## rotor.input File

- Articulated rotors need only a subset of the data (manual defines variables)

```

Rotors Uinf/Uref Write Soln Force Ref Momment Ref ! Below we set Uref = Utip
1 0.245 1500 1.0 1.0 ! Adv Ratio = Uinf/Utip
 ! So here Uinf/Uref = AR

=== Main Rotor =====
Rotor Type Load Type # Radial # Normal Tip Weight
1 1 50 180 0.0
X0_rotor Y0_rotor Z0_rotor phi1 phi2 phi3
0.0 0.0 0.0 0.00 0.0 0.00
Utip/Uref ThrustCoff TorqueCoff psi0 PitchHinge DirRot
1.0 0.0064 0.00 0.0 0.0466 0
Blades TipRadius RootRadius BladeChord FlapHinge LagHinge
4 26.8330 2.6666 1.741 0.0466 0.0466
LiftSlope alpha, I=0 cd0 cd1 cd2
6.28 0.00 0.002 0.00 0.00
CL_max CL_min CD_max CD_min Swirl
1.50 -1.50 1.50 -1.50 0
Theta0 ThetaTwist Thetals Thetalc Pitch-Flap
0.0 0.00 0.0 0.0 0.00
FlapHar Beta0 Betals Betalc
0 0.0 0.0 0.0
Beta2s Beta2c Beta3s Beta3c
0.0 0.0 0.0 0.0
LagHar Delta0 Deltals Deltalc
0 0.0 0.0 0.0
Delta2s Delta2c Delta3s Delta3c
0.0 0.0 0.0 0.0

```

Key:  
 Required for rigid and elastic  
 Required for untrimmed rigid  
 Unused (must have a value)



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



17

## Input For Articulated-Blade Simulations (1/2)

- Except as noted, inputs pertain to both untrimmed/rigid-blades and trimmed/elastic blades
- Run as time-dependent, so will need to set time step as per slide 13
- Required additional `fun3d.nml` input

```

&global
 moving_grid = .true.
 slice_freq = 1 (optional if rigid untrimmed)
/
&rotor_data
 overset_rotor = .true.
/
&overset_data
 overset_flag = .true.
 dci_on_the_fly = .true. (potentially optional if rigid)
 dci_period = 360 (assuming 1 deg. per time step)
 reuse_existing_dci = .true.
/

```



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



18

## Input For Articulated-Blade Simulations (2/2)

- The `moving_body.input` file is somewhat simplified since much of the motion description is handled by `rotor.input` – all we need do is define the moving bodies and provide the SUGGAR++ xml file if required

```
&body_definitions
 n_moving_bodies = 4 (e.g. for 4-bladed rotor)
 body_name(1) = 'rotor1_blade1' (same as in xml file)
 n_defining_bndry(1) = 2
 defining_bndry(1,1) = 3
 defining_bndry(1,2) = 4
 mesh_movement(1) = 'rigid+deform' (or just 'rigid' for
 for rigid blade case)

 ... (etc. for blades 2-4)
/
&composite_overset_mesh
 input_xml_file = "Input.xml_0" (potentially optional if rigid
 and have precomputed dci)
/
```

- Note: `motion_driver` not set in `&body_definitions` (in contrast to any other moving grid case); also *no* `&forced_motion` input



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



19

## CAMRAD Considerations

- User must set up basic CAMRAD II scripts; the `RUN_LOOSE_COUPLING` script provided with FUN3D requires 3 distinct, but related CAMRAD scripts
  - `basename_ref.scr`
    - Used to generate the reference motion data used by CAMRAD
    - Set this file to use rigid blades; zero collective/cyclic; no trim
  - `basename_0.scr`
    - Used for coupling/trim cycle “0”
    - Set up for elastic blades with trim; use CAMRAD aerodynamics exclusively (no delta airloads input); simplest aero model will suffice
  - `basename_n.scr`
    - Used for all subsequent coupling/trim cycles
    - Set up for elastic blades with trim; use same simple CAMRAD aerodynamics but now with delta airloads input



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



20

## Blade Surface “Slicing”

- Boundary surface (rotor blade) slicing is *required* for coupled CFD/CSD simulations; also useful for rigid-blade cases - this is what generates the data in **rotor\_1.onerev.txt**

```
$slice_data
 replicate_all_bodies = .true. ! do the following the same on all blades
 output_sectional_forces = .false. ! just lots of data we usually don't need
 tecplot_slice_output = .false. ! ditto
 slice_x(1) = .true., ! x=const slice - in original blade coords
 nslices = -178, ! no. slices; "-" means give start and delta
 slice_location(1) = 2.8175, ! x-location to slice (starting slice)
 slice_increment = .13416666666 ! delta slice location each successive slice
 n_bndrys_to_slice(1) = 1, ! 1 bndry to search
 bndrys_to_slice(1,1) = 2, ! indicies:(slice,bdry) lumping made life easy
 slice_frame(1) = 'rotor1_blade1', ! ref. frame in which to slice - use body name
 te_def(1) = 20, ! look for 2 corners in 20 aft-most segments
 le_def(1) = 30, ! search 30 fwd-most pts for one most distant from TE
 chord_dir(1) = -1, ! Recall goofy original blade coord system
/
```

- Note: “slicing” useful for applications other than rotorcraft; see website



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



21

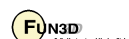
## Untrimmed Rigid-Blade Simulations

- Overview of the basic steps
  1. Prepare rotor blade and fuselage grids, with proper axis orientation
  2. Set up the **rotor.input** file based on flight conditions
  3. Run the **dci\_gen** utility to create a composite mesh and initial dci data
  4. Set up **fun3d.nml** and **moving\_body.input** files
  5. Optionally set up the **&slice\_data** namelist in the **fun3d.nml** file
  6. Run the solver; the number of time steps required is case dependent – usually at least 3 revs for rigid blades



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



22

## Trimmed, Elastic-Blade Simulations

- Overview of the basic steps; steps 1-4 are the same as for the untrimmed rigid-blade case; use of CAMRAD is assumed
- 5. Set up the `&slice_data` namelist; set `slice_freq = 1` *not optional*
- 6. Set up the 3 CAMRAD run-script templates
- 7. Set up the `RUN_LOOSE_COUPLING` run script (a c-shell script geared to PBS environments); user-set data is near the top – sections 1 and 2
- 8. Set up the `fun3d.nml_initial` and `fun3d.nml_restart` files used by the run script; typically set the time steps in the initial file to cover 2 revs, and  $2/N_{\text{blade}}$  revs in restart version
- 9. Before using the run script make sure all items it needs are in place; script checks for missing items, but it gets old having to keep restarting because you forgot something!
- 10. Number of coupling cycles required for trim will vary, but 8-10 is typical for low-moderate thrust levels; high thrust cases near thrust boundary may require 10-15; user judges acceptable convergence



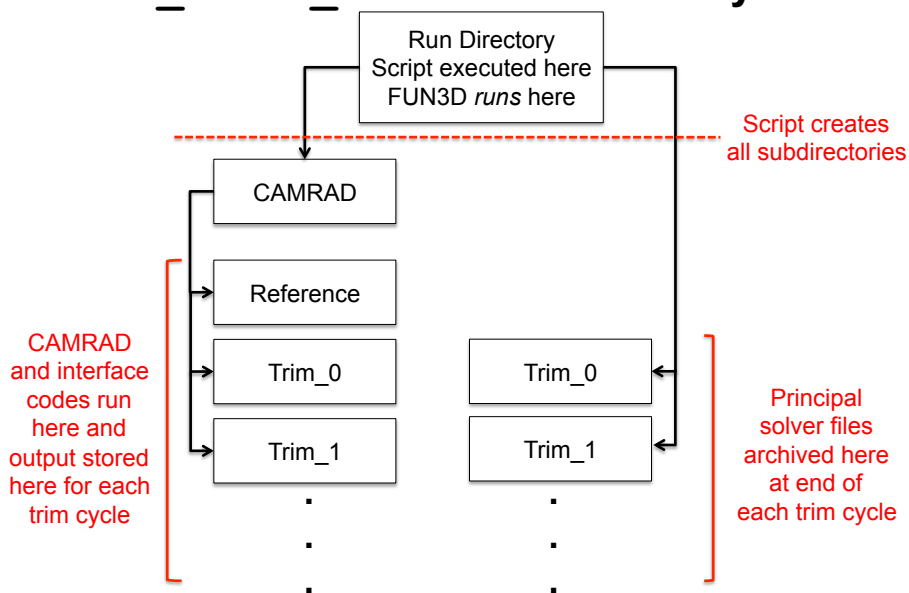
<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



23

## RUN\_LOOSE\_COUPLING Directory Tree

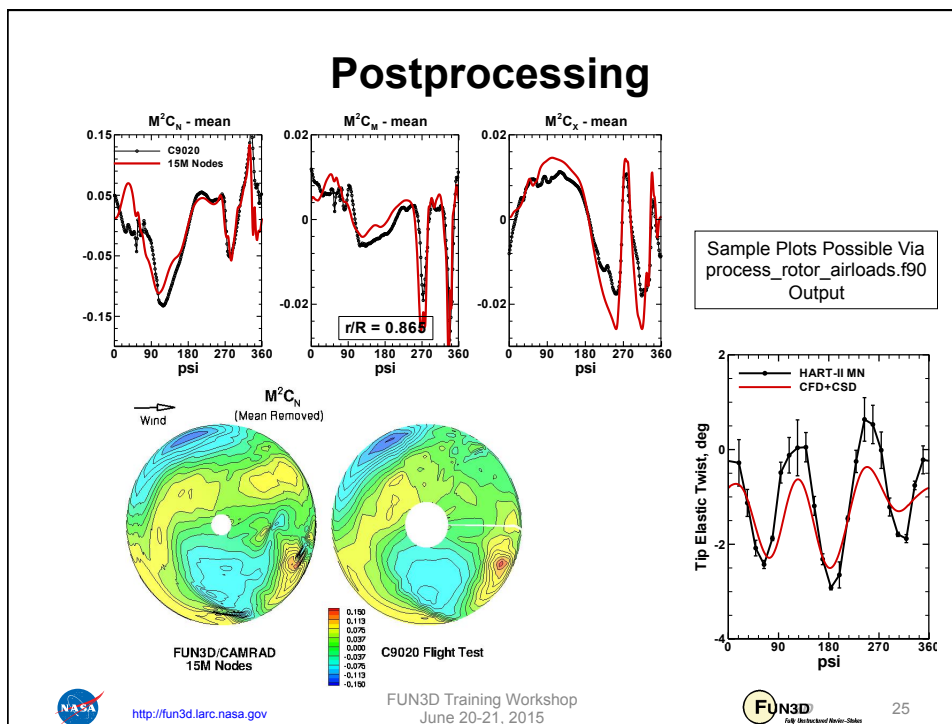


<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



24



## Non-Inertial Reference Frame (1/2)

- For **isolated, rigid** an improvement in solution efficiency may be obtained by transforming to a coordinate system that rotates with the rotor
- FUN3D implements a very limited subset of possible non-inertial frames:
  - Constant rotation rate
  - Free-stream flow limited to
    - Quiescent (e.g. rotor in hover)
    - Flow aligned with axis of rotor (e.g. ascending/descending rotor; prop in forward flight at 0 AoA)
- In this noninertial rotating frame, the flow is assumed steady
- Can be used in conjunction with overset grids to allow pitch/collective changes to rotor without re-gridding
- The noninertial capability has other limited applications in addition to rotors – e.g. aircraft in a steady loop

## Non-Inertial Reference Frame (2/2)

- `fun3d.nml` input for non-inertial frame solutions (example for rotor spinning about z-axis)

```
&noninertial_reference_frame
 noninertial = .true.
 rotation_center_x = 0.0 !rotation axis passes through this pt.
 rotation_center_y = 0.0
 rotation_center_z = 0.0
 rotation_rate_x = 0.0
 rotation_rate_y = 0.0
 rotation_rate_z = 0.2
/
```

- The nondimensional rotation rate is determined as shown on slide 11
- Flow-visualization output (boundary, volume, sampling) will be relative to the non-inertial frame

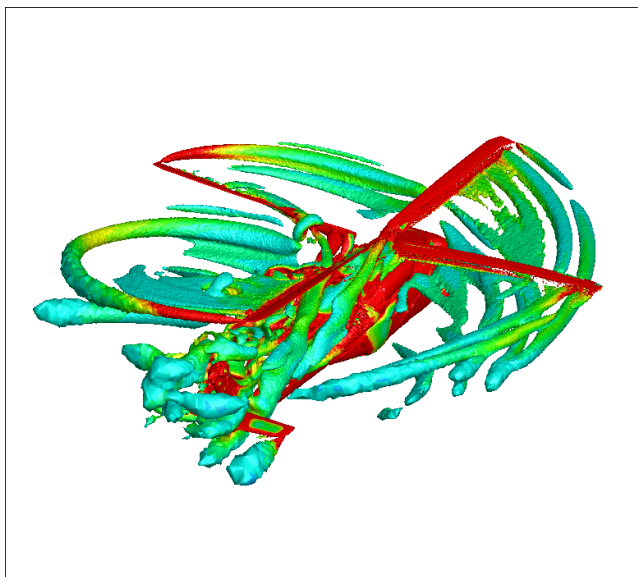


<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



27



<http://fun3d.larc.nasa.gov>

FUN3D Training Workshop  
June 20-21, 2015



28