Ongoing Research Into Numerical Simulation of Fluid Flows Utilizing Software Development Practices

FUN3D Software Development Team

NASA, Hampton, Virginia

ACDL Seminar

by mikePark@MIT.Edu Mike.Park@NASA.Gov

24 September 2004

FUN3D

Originated by Kyle Anderson, Eric Nielsen, and others

Extended by High Energy Flow Solver Synthesis (HEFSS) effort

An element of the Fast Adaptive AeroSpace Tools (FAAST) project

Detailed in *Breakthroughs in large-scale computational simulation and design* (NASA/TM 2002-211747)

Unstructured-grid analysis and design across speed range:
Incompressible, compressible, hyper sonic reacting gas

FUN3D	unstructured-grid, incompressible/compressible
LAURA	structured-grid, external hypersonics
VULCAN	structured-grid, internal hypersonics

Why?

Multidisciplinary problems require multiple discipline experts, a large infrastructure, and standard interfaces

Reduce time from concept to application for vehicles and algorithms

Mobility to respond to unforeseen challenges and increase software lifespan

Research capabilities in a “production” code

Infrastructure to evaluate algorithms on large problems

Flexibility for implementing research algorithms

Stability to suit time-sensitive application needs and to release to outside customers

Avoid being encumbered by high-ceremony software development process

Software Versioning System (Control)

Often overlooked or under emphasized

Zeroth principle of software engineering

Learning to work with it and not against it is key to team programming (glue)

Safety net

Large impact on “Truck Number”

Convenient for accounts on multiple machines

Required for automated testing

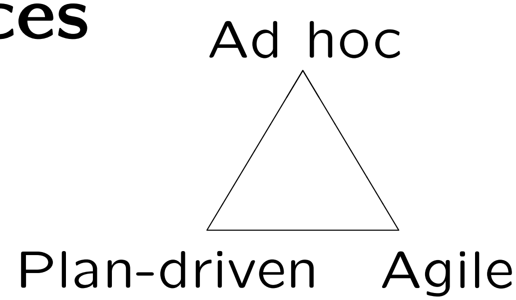
Not just for software anymore

- homework, presentations, configuration files, home accounts

CVS - `https://www.cvshome.org/`

Subversion - `http://subversion.tigris.org/`

Software development practices



"Kleb Triangle"

Ad hoc

- "Code and Fix"

Plan-driven

- Predictive, "Big up front design"
- Delivering to the original contract
- Capability Maturity Model (CMM), CMMI

Agile

- Adaptive, "Evolutionary design"
- Recognizes software development an empirical process that can not always be defined
- Extreme Programming

The Agile Manifesto values

individuals and interactions

over processes and tools

working software

over comprehensive documentation

responding to change

over following a plan

customer collaboration

over contract negotiation

Extreme Programming values

communication

simplicity

feedback

courage

Extreme Programming Practices

Sustainable pace productivity does not increase with hours worked.

Metaphor guide all development with a simple shared story of how the whole system works.

Coding standard write all code in accordance with rules emphasizing communication through the code.

Collective ownership anyone can change any code anywhere in the system at any time.

Continuous integration integrate and build the system many times a day.

Small releases release new versions on a very short cycle.

Extreme Programming Practices (concluded)

Test-driven development any program feature without an automated test simply does not exist.

Refactoring restructure the system without changing its behavior.

Simple design system should be designed as simply as possible at any given moment.

Pair programming two programmers work together at one computer on the same task.

On-site customer include a real, live user on the team.

Planning game combine business priorities and technical estimates to determine scope of next release.

FUN3D Development

Sustainable pace work ∼40 hour weeks.

Metaphor engineering and scientific vocabulary (ρ, u, v, w).

Coding standard published to aid portability, automated parsing, and collective ownership.

Collective ownership routinely fix minor bugs or extend methods created by other people. Anyone is allowed to modify any file at anytime through CVS.

Continuous integration very slow: Linux builds every 2-3 hours, SGI builds every 8-9 hours.

Small releases application members of the team use (CVS), formally 2-3 times a year

FUN3D Development (concluded)

Test-driven development limited use in flow solver, extensive use in scripting and grid adaptation.

Refactoring done only when necessary, extremely difficult, painful, and nerve-racking without unit tests.

Simple design born as a result of refactoring and pair programming.

Pair programming limited to mostly debugging, knowledge transfer; impeded by scheduling conflicts.

On-site customer research: we are our own customers?

Planning game comes more naturally in pair programming scheduling.

Communication

Collocation

Email list

WikiWikiWeb – `http://c2.com/cgi/wiki?WelcomeVisitors`

Scrum status meetings

What they **did** since last meeting

What they will **do** by next meeting

What got **in the way** (impediments)

Quick and efficient meeting style

Reduces the worst management sin (wasting people's time)

The impediments is the often the hardest to express, but the most important.

Software Testing

All of these can (and should) be automated!

Programmer's I want a function that adds vectors, does $f([1,2],[3,4])$ return $[4,6]$? (unit tests)

Integration Does my whole system compile and work together?

Regression My code gave answer x yesterday, does it give answer x today?

Verification My code is supposed to be second-order accurate in space. What happens when I change the element size?

Validation Does my code give the same answer as a wind tunnel or fight test measurement?

Unit Testing Frameworks

Goals

- Interface must allow for the easy creation and management of tests
- Minimal additional effort over writing the actual code (benefit–cost)
- Enable programmers to experience the benefits of test-first programming as soon as possible
- Legible as documentation

Flavors

- **`http://c2.com/cgi/wiki?TestingFramework`**
- Full featured (scripting languages)
- Minimalistic (four lines of code)
- Wrap code to utilize scripting language framework
- "Roll your own"

Unit Testing and Test First Programming

Seems trivial at first

Hard to imagine benefit until the first major refactoring or code simplification is experienced

Gains power as the number of tests and their coverage increases

Your own custom debugger

Provides a clear completion to an implementation task

Code with a failing test is much easier to fix or extend

Inventing the tests required is generally harder

Code that is easy to test is often simpler and easier to read, understand, and extend

Creating tests brings the design to the forefront; design is difficult, but it is easiest in small increments

Discretization error is a major problem

- AIAA Drag Prediction Workshop
  - A large number of people applied a large number of codes to a single transport configuration
  - Large spread in results (largest may be programming errors)
  - Grid converged answers where not demonstrated even for large grids (asymptotic range)
  - Discretization error explicitly identified in reports
- Multi element high lift and sonic boom calculations
- Preventing the characterization of modeling errors (turbulence models)
- It is often combated by specifying local grid resolutions by hand (requires expert with past experience with similar problems)
- Local error estimates have been useful for adaptation but often fail for problems that are strongly nonlinear or when transported error overcomes the solution

Adjoint solution

Linearized flow residual and output quantity at a flow state

Efficient method for computing derivatives and design

Existing NASA Langley technology for 3D turbulent design problems

Shows the linearized impact of a equation source term (error) on an output function

Indicates the global impact of local errors when combined with local error estimates

Error Estimation and Adaptation

Improve function calculation by predicting and correcting error

Combining flow and adjoint problems

Adapt discretization to improve (not reduce) correction

- Error estimate can be computed to high accuracy, but it is a linear correction

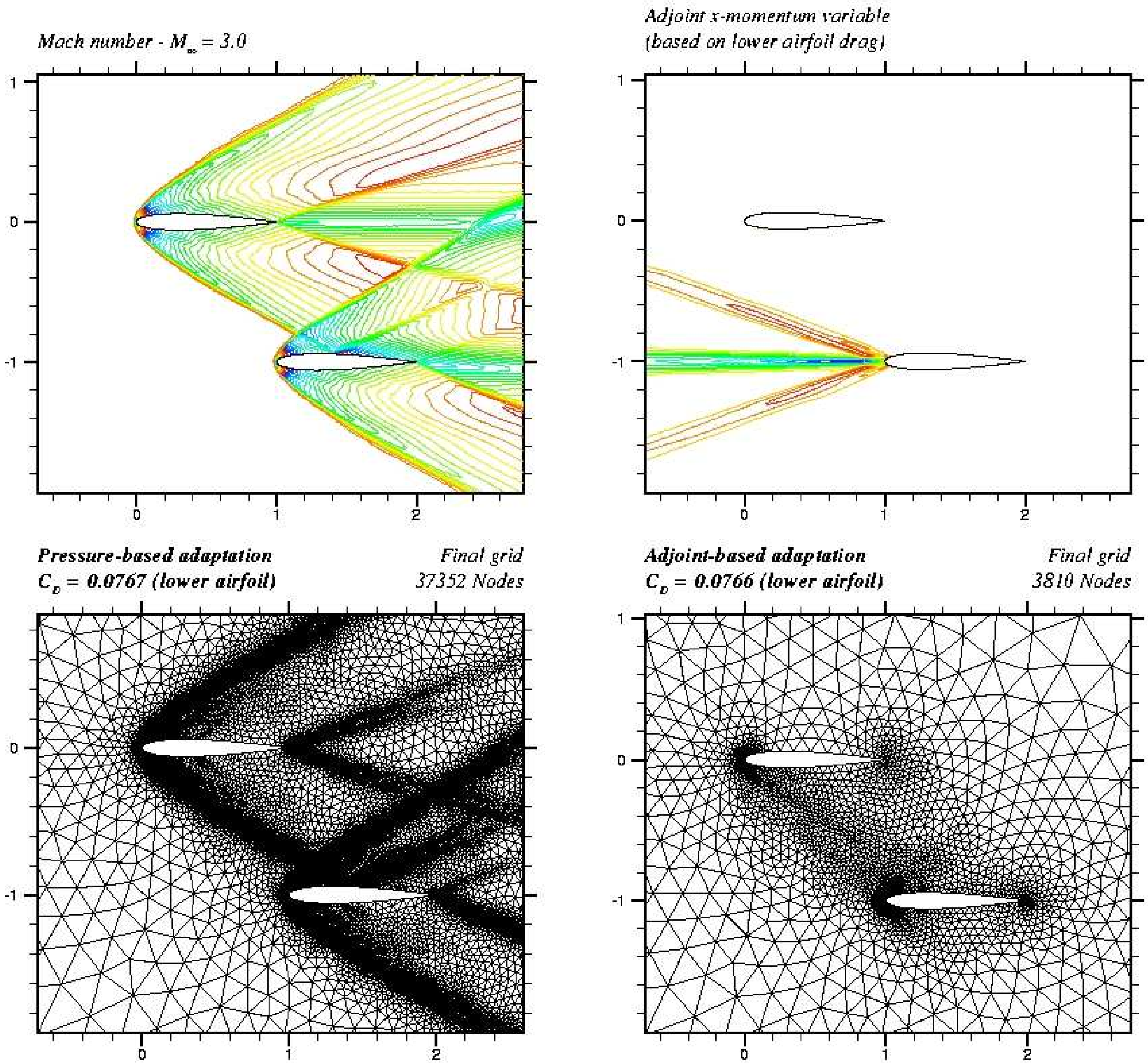
Based on the 2-dimensional (2D) work of

- Venditti and Darmofal (MIT)
- Müller and Giles

Intended to avoid manually specifying grid resolution to enable design

Requested metric is a Mach Hessian scaled by the adjoint error estimate uncertainty

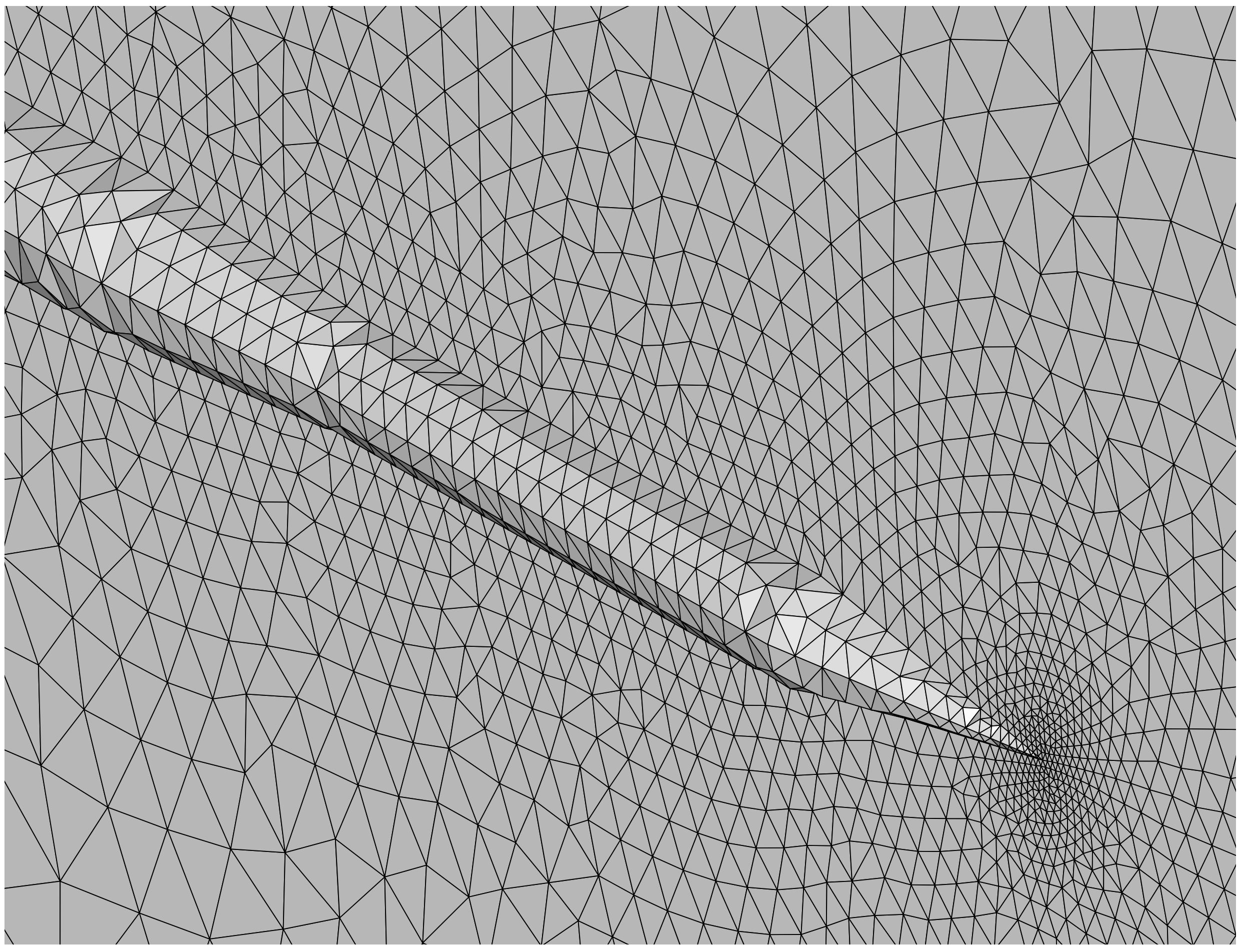
Venditti and Darmofal Mach 3.0 Biplane



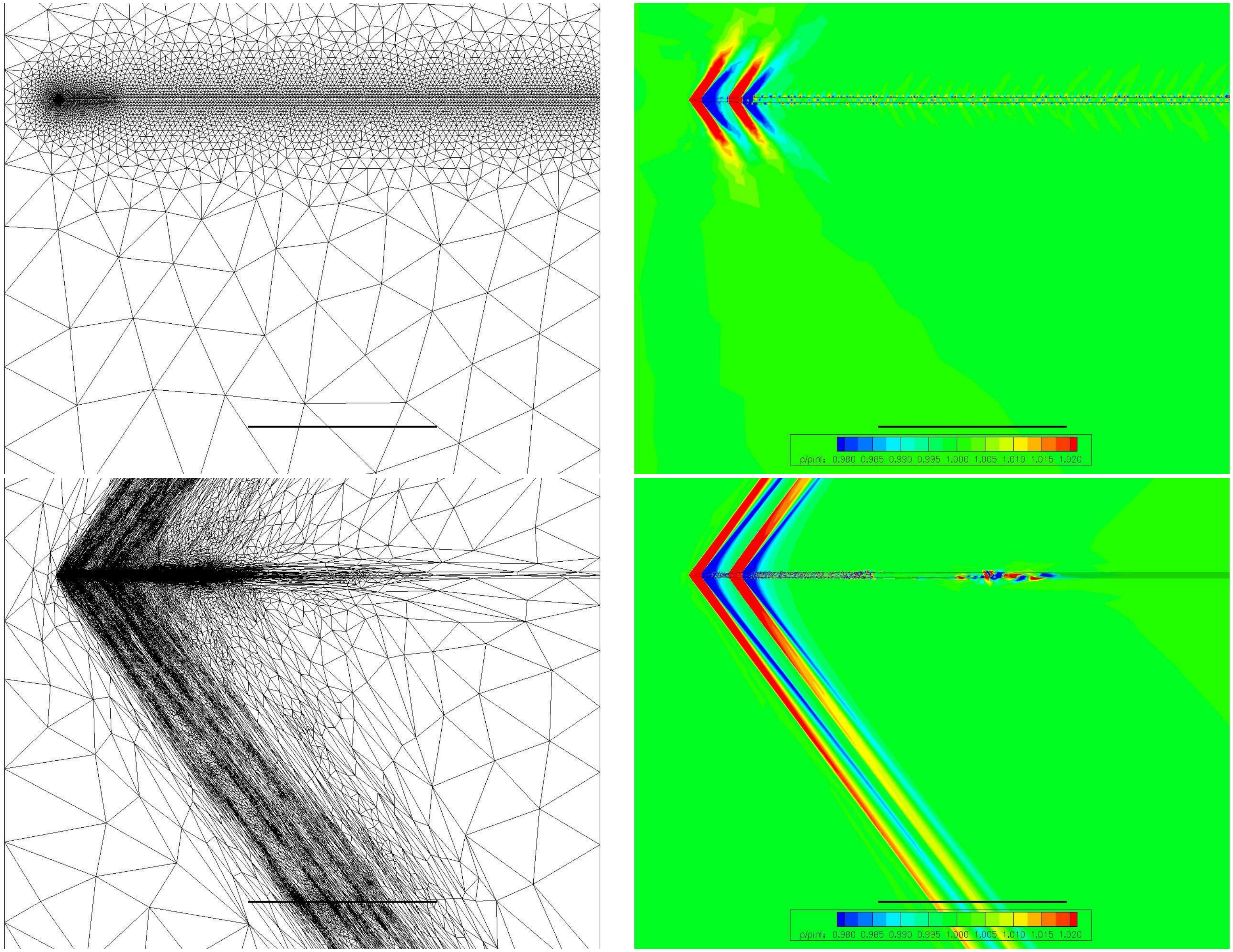
Mach 1.26 Double Cone Shock Propagation

Wind tunnel data from 1965 Technical note NASA TN D-3103

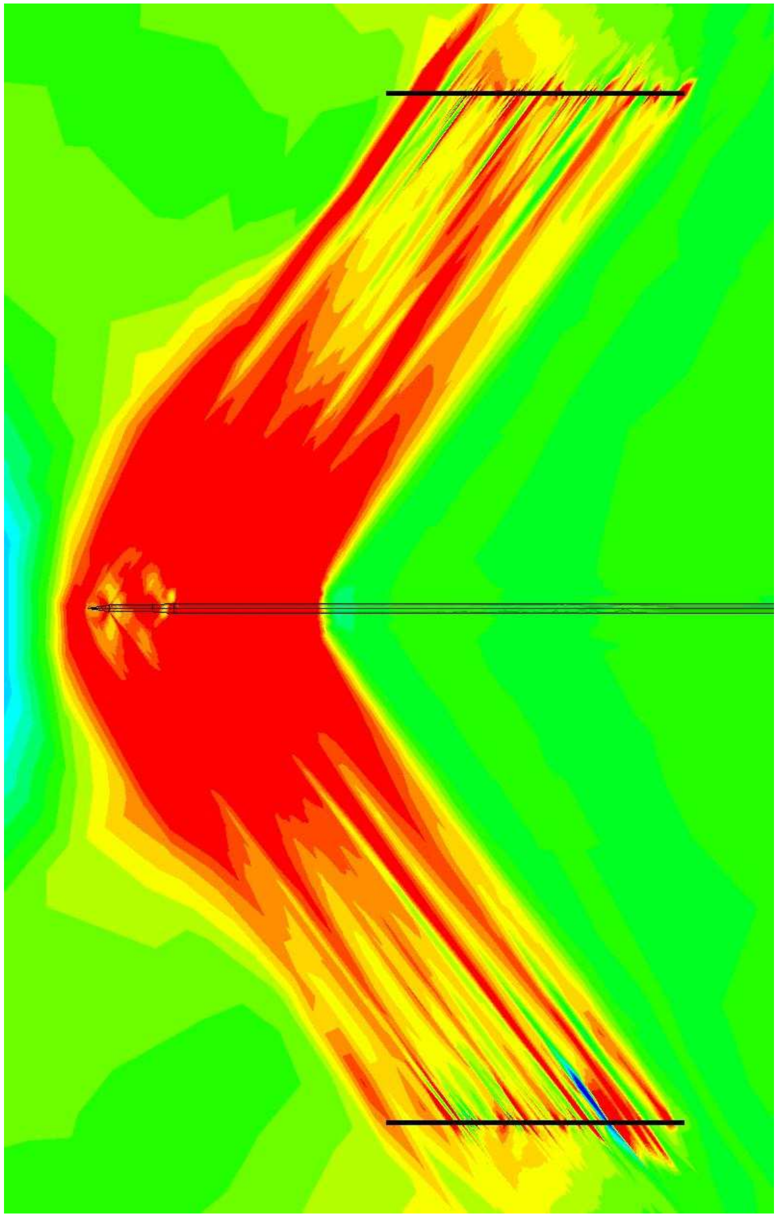
Cost is the integral of pressure on a cylinder with 6 body length radius



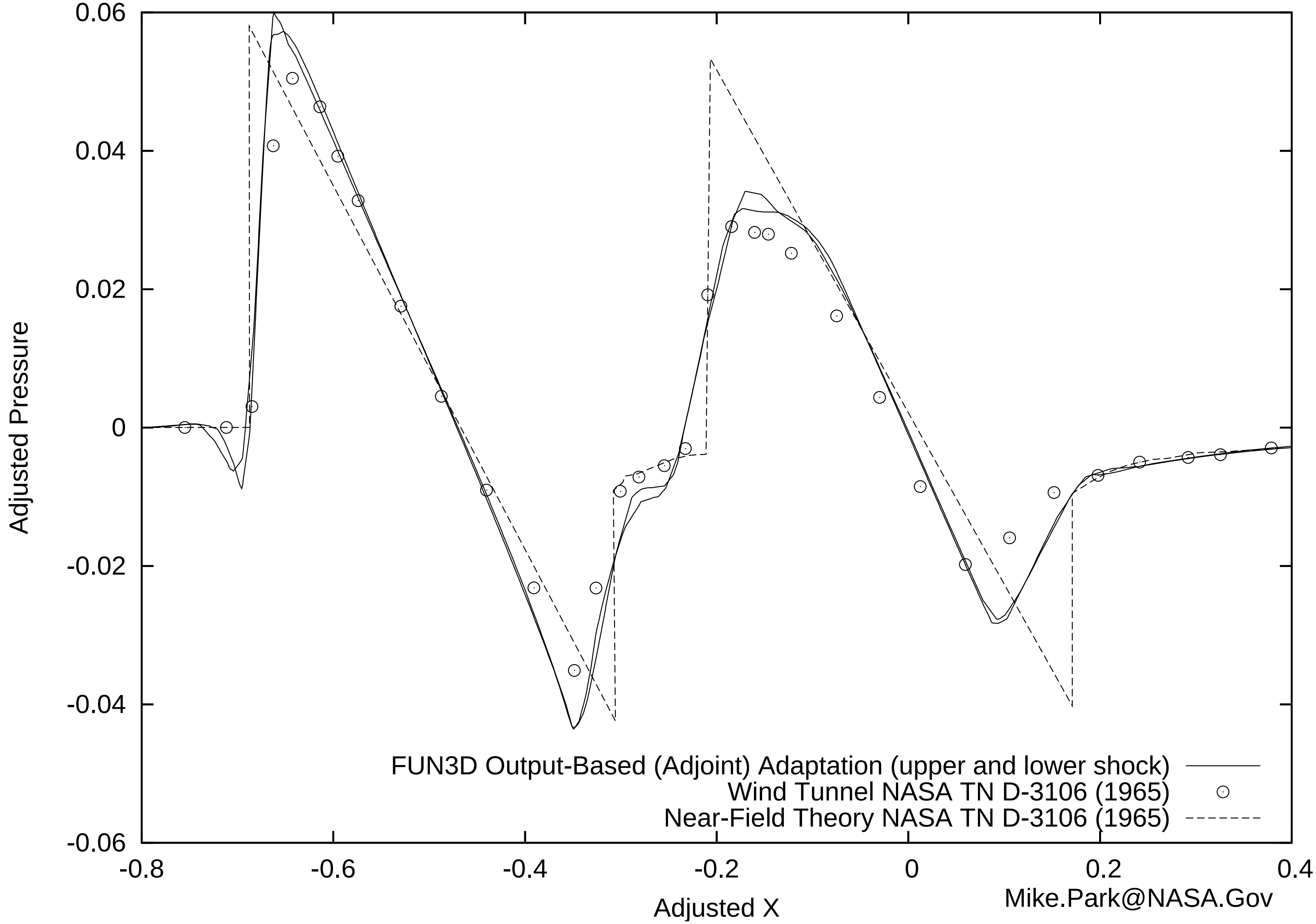
Double Cone Shock Propagation Mach 1.26



Double Cone Shock Propagation Mach 1.26 Adjoint Variable

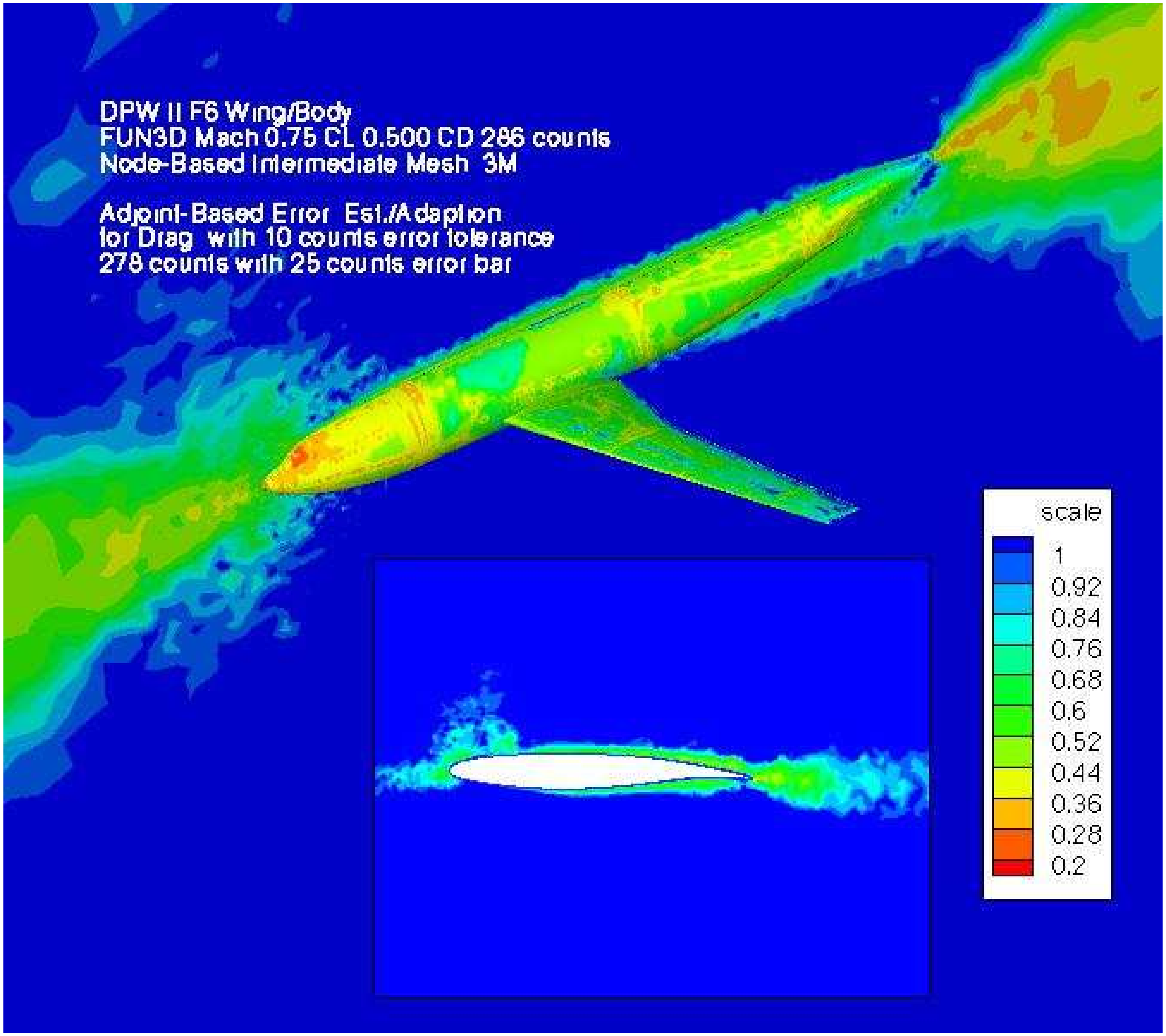


Double Cone Shock Propagation Pressure at 6 Body Lengths



Mike.Park@NASA.Gov

Drag Prediction Workshop DLR F-6 (Beth Lee-Rausch)



Adaptation Mechanics

Limiting process for combining design and output based adaptation

Needs to align grid with strongly anisotropic regions in the solution

Improved robustness may result in never needing to look at the grid

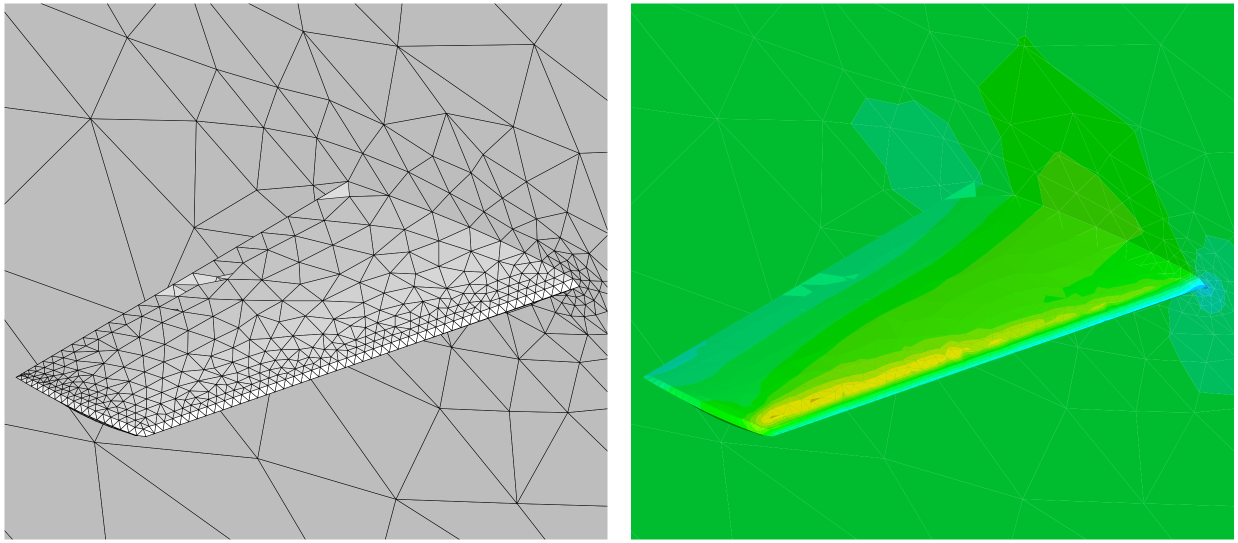
Compatible with flow solver grid quality requirements

Should work seamlessly with the flow solver and error estimation process

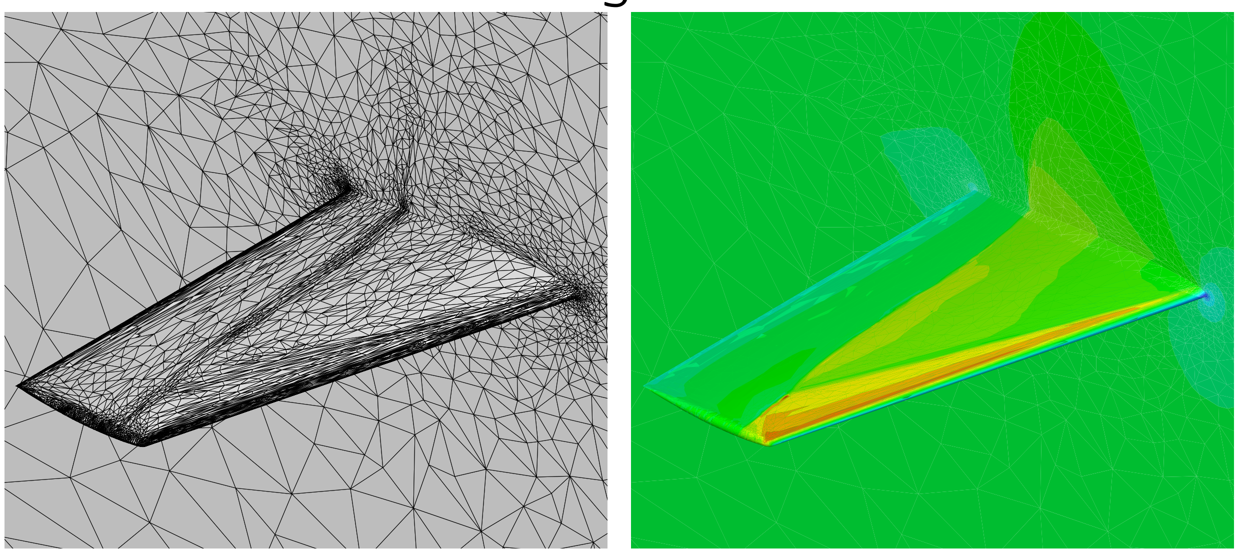
Anisotropic Adaptation Mechanics – “refine” Library

- Written in C, wrapped with Ruby scripting language
- Test-first development (unit tests)
- GNU Autotools
- Focus on building reusable infrastructure
  - Refinement and coarsening
  - Edge/face swapping
  - Node smoothing and untangling
  - Global grid movement
  - Projection to CAD (CAPRI)
- Parallelized and coupled to the FUN3D (load balancing)
- Focus of current research

ONERA M-6 Wing Inviscid Drag Adaptation Mach 0.84

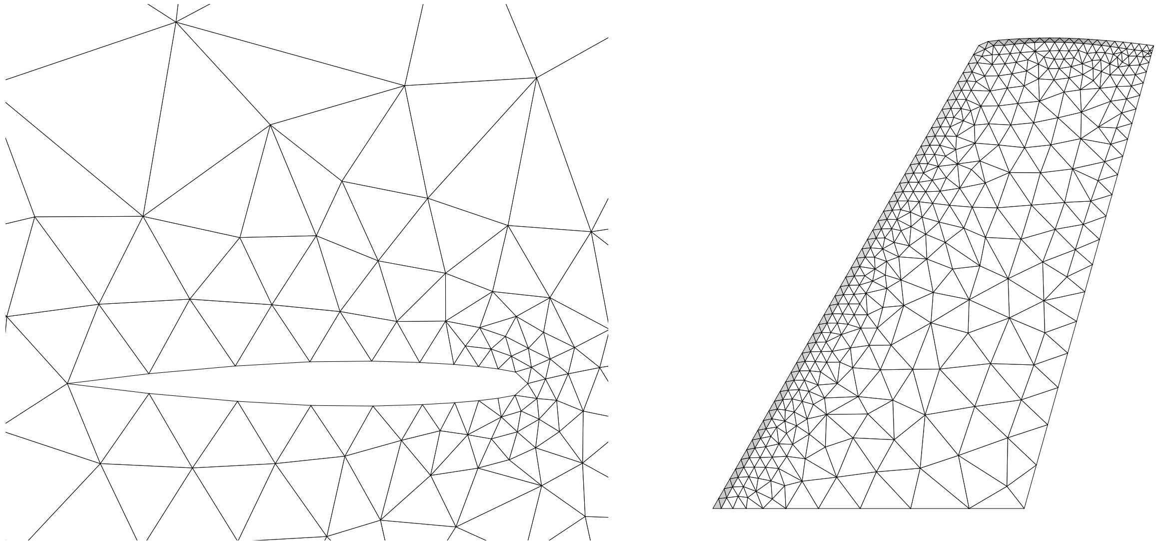


Original

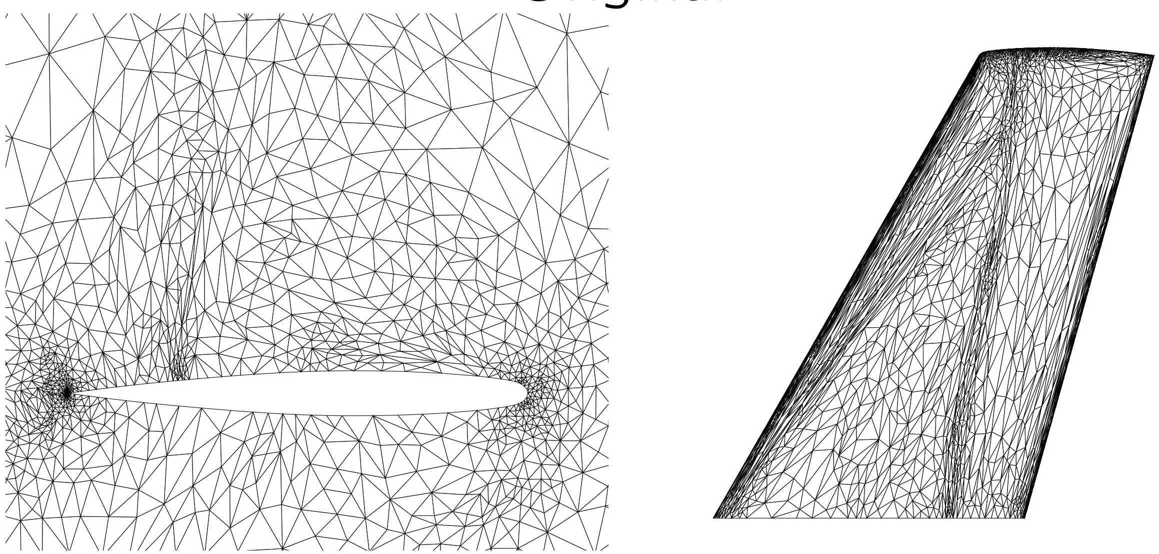


Final

ONERA M-6 Wing Inviscid Drag Adaptation Mach 0.84



Original



Final

Software development Practices

Software version control

Agile software development practices

Types of software testing

Discretization Error

Major issue

Estimation and control by adaptation

Adaptation Mechanics

Limiting factor on applying output based adaptation to design and turbulent analysis