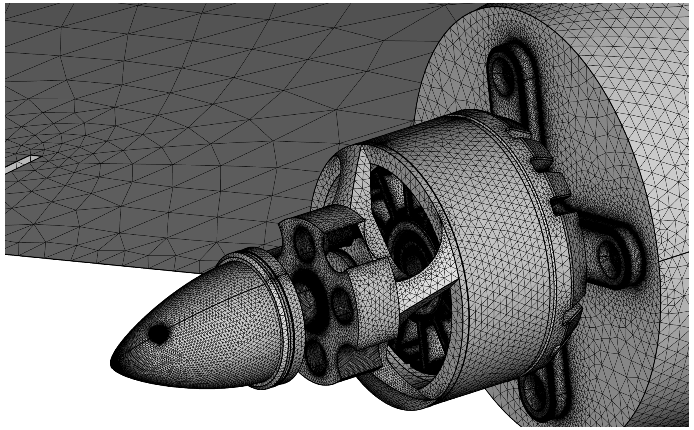
Anisotropic Mesh Adaptation with the (Py)AMG Library

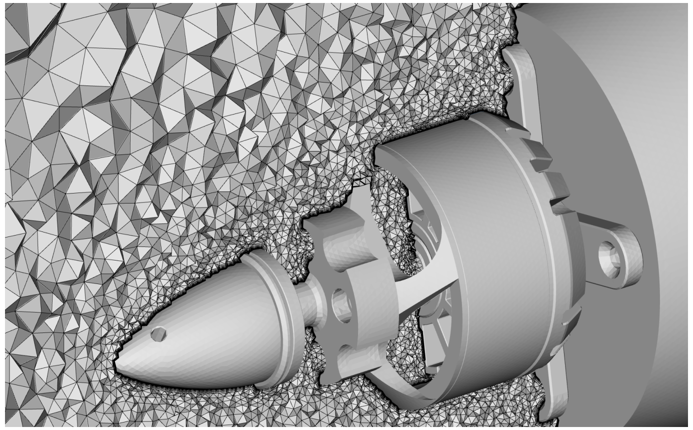
Adrien Loseille (INRIA),
Victorien Menier (Stanford), Brian Munguía (Stanford)
And
Juan J. Alonso (Stanford)

3rd Annual SU2 Developers Meeting

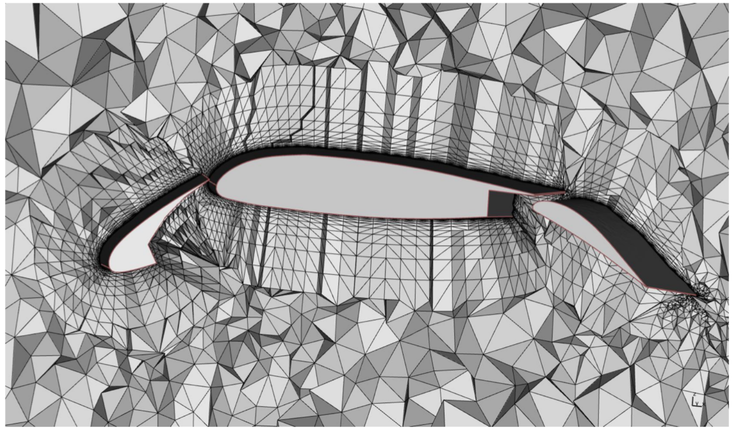
Research Interests of the Gamma3 Team



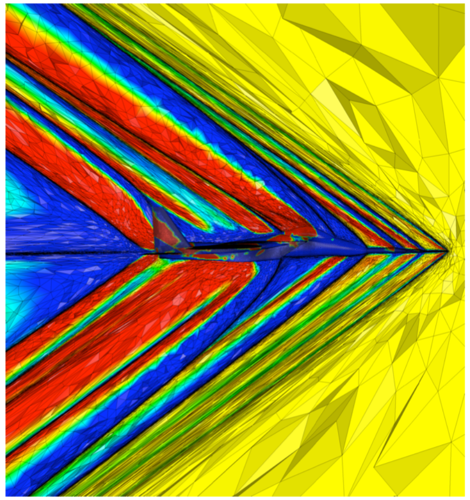
Surface meshing
(triangular, hexahedral)
(software: BLSURF, Hexotic)

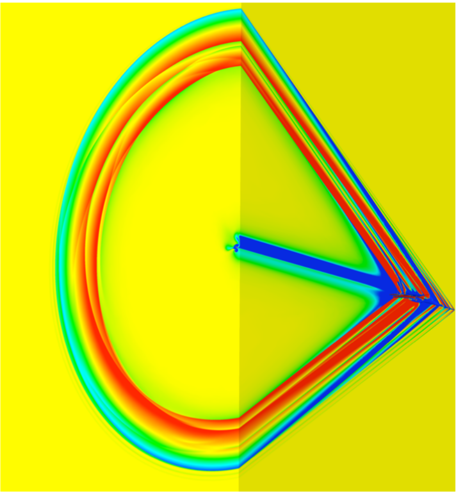


Volume meshing
(Software: GHS3D, Hexotic)



Boundary layer meshing
(AMG_BL, Bloom)





Adaptivity : Error estimates, Anisotropic mesh generation

INRIA/Stanford collaboration

- Informal collaboration for a long time (10 years)
- 2014 : Collaboration of 1st AIAA Sonic Boom Workshop
  - First anisotropic computations with SU2/AMG, joint work with ONERA
  - Inviscid runs
- 2016 : Internal use during V. Menier's post doc

→ First discussions of exposing an interface for SU2

Outline

- Background on anisotropic mesh adaptation
  - Metric-based mesh generation
  - Error estimates
  - Unique cavity-based operators
  - Numerical examples : RANS Adaptation

- PyAMG interface and SU2
  - Python bindings
  - Numerical adaptive examples

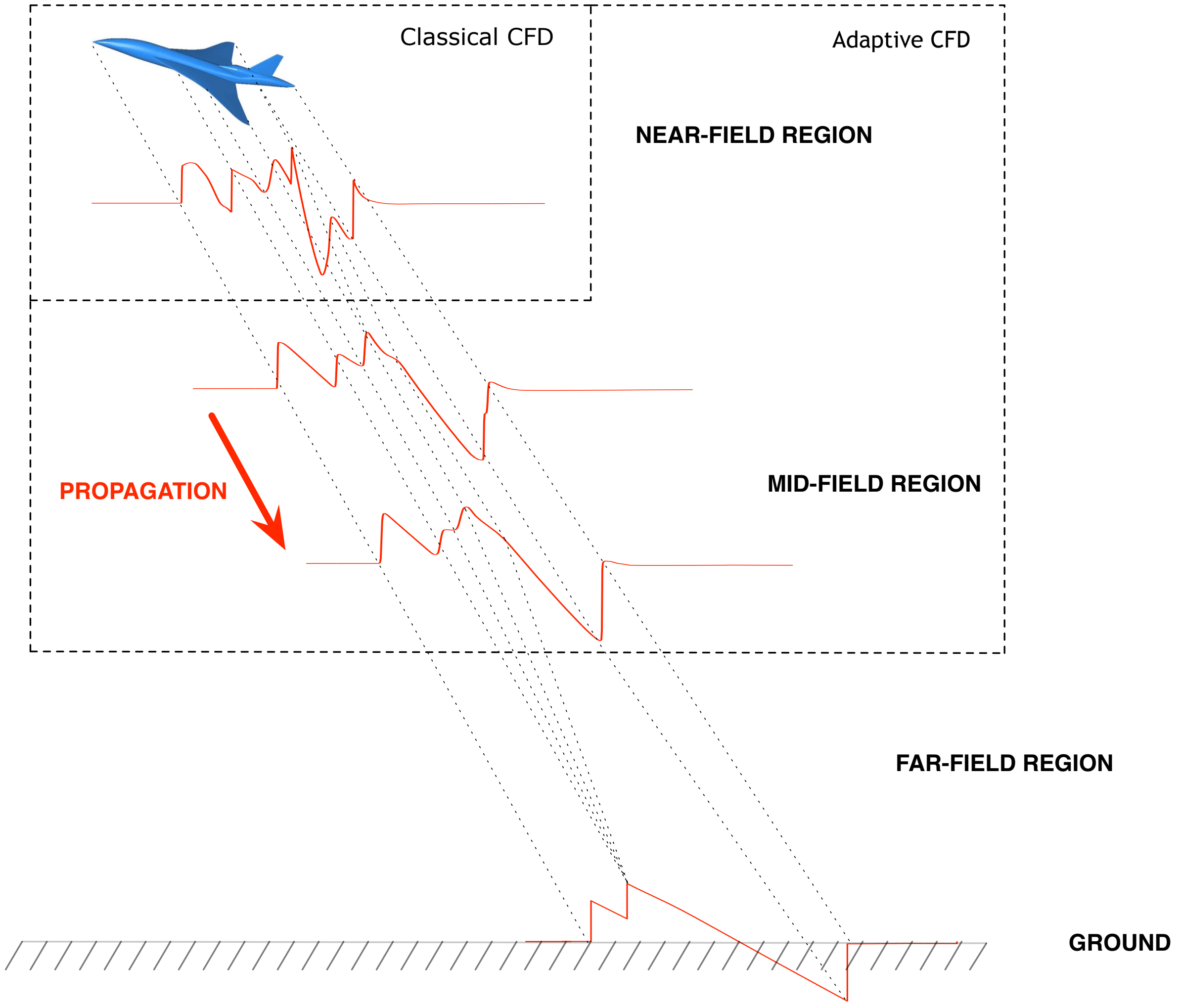
Motivations

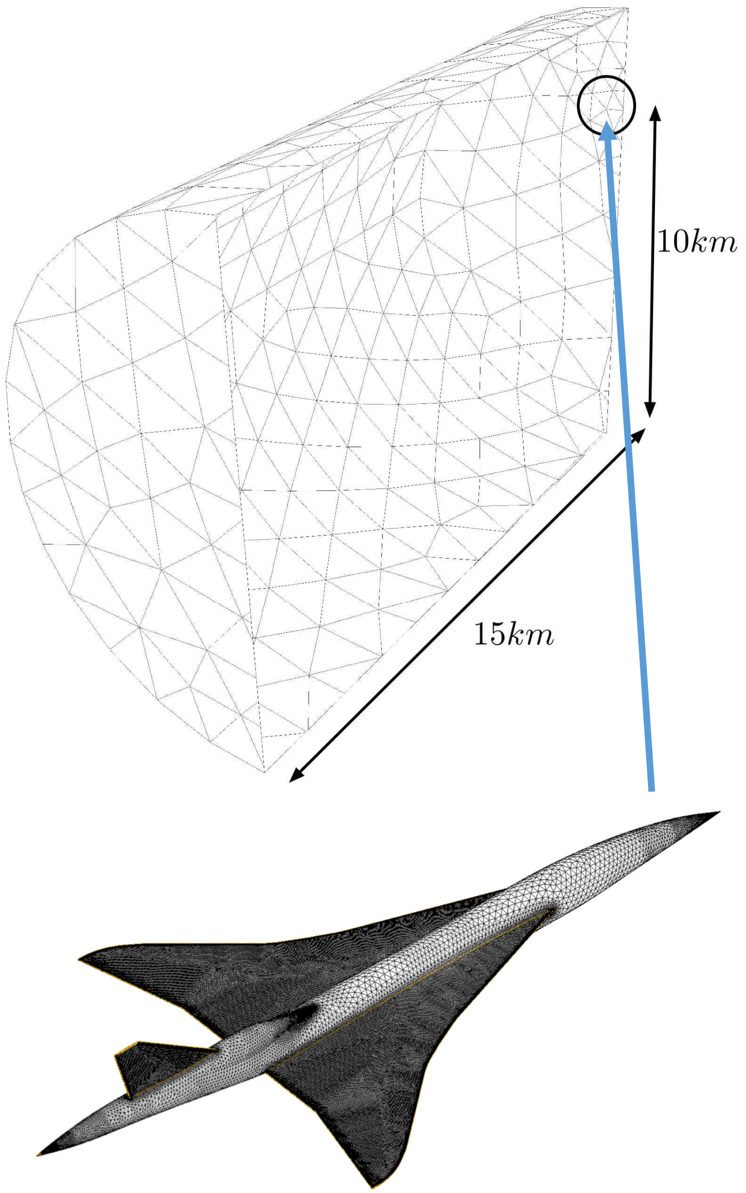
- Physical phenomena have strong anisotropic components
- Uniform meshes are not optimal in term of sizes and directions

- Anisotropic mesh adaptation is a way to optimize the ratio

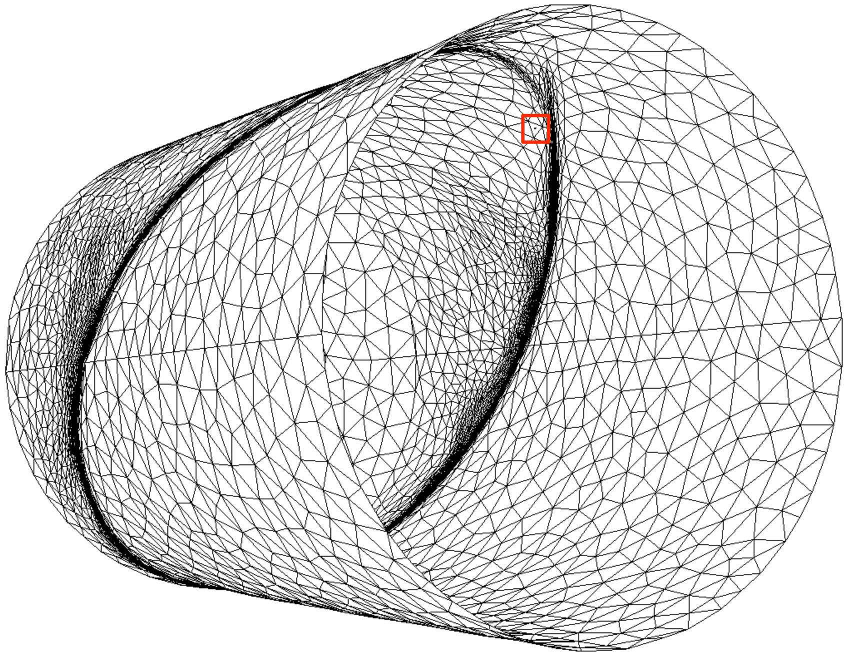
CPU time versus accuracy

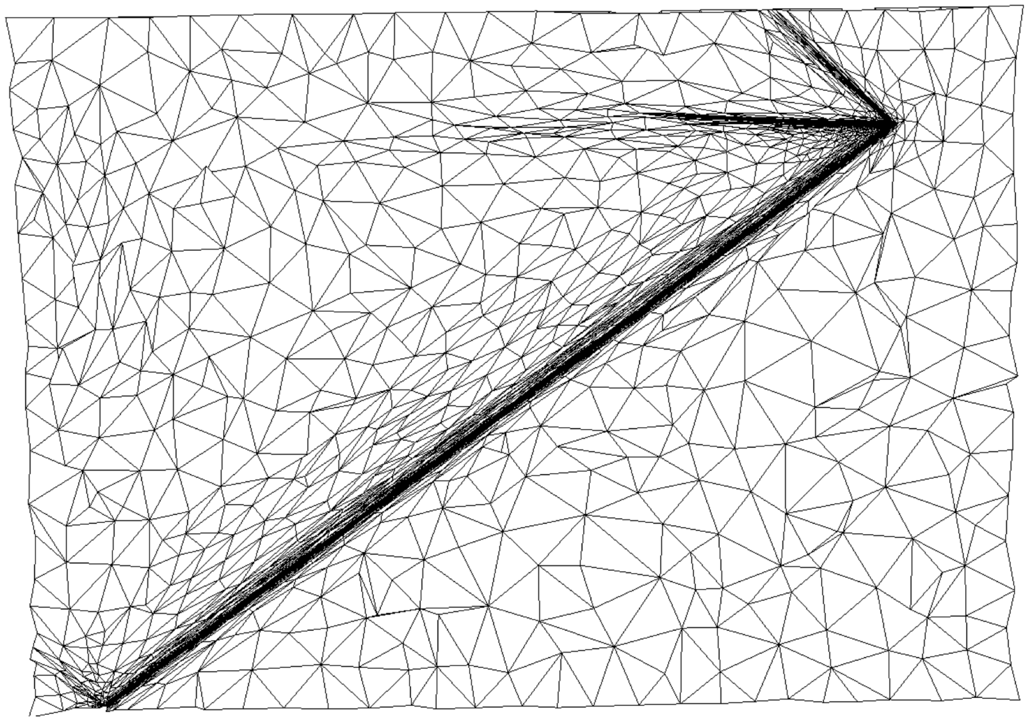
Example : Direct sonic boom



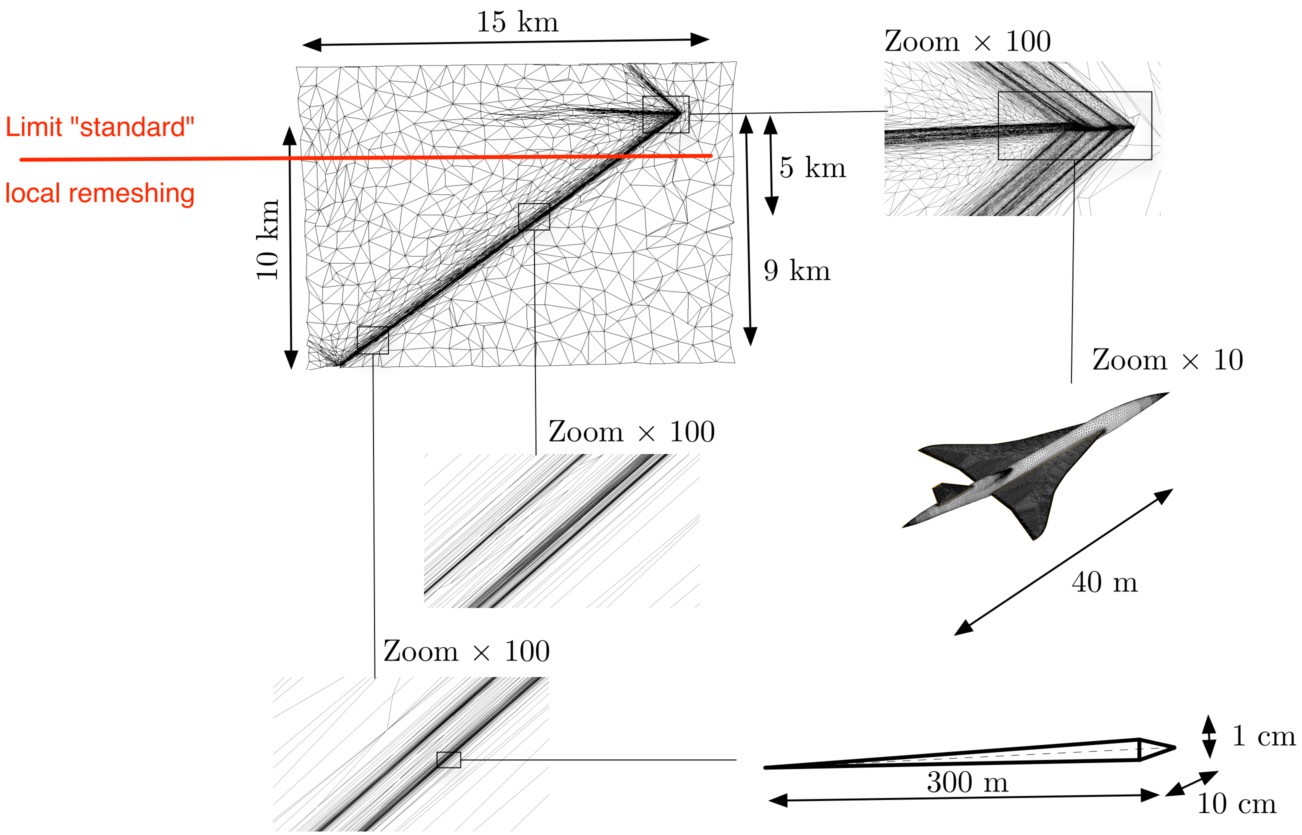


Example : Direct sonic boom





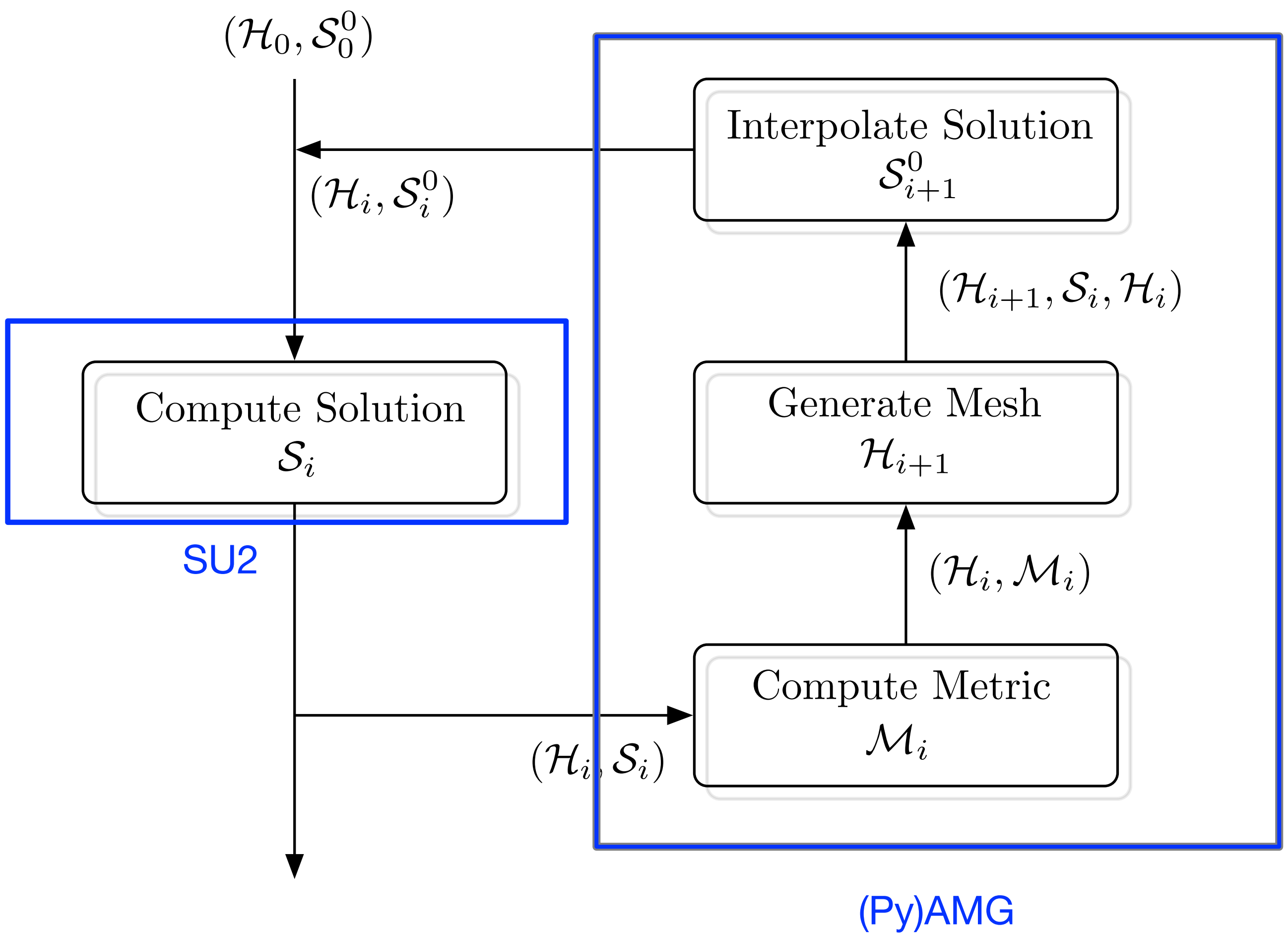
- Initial mesh 415 535 vertices
- Volume ratio [5.e^{-11} 4.7e^{10}]
- Size ratio 10^9
- Final adapted 3 299 676 vertices

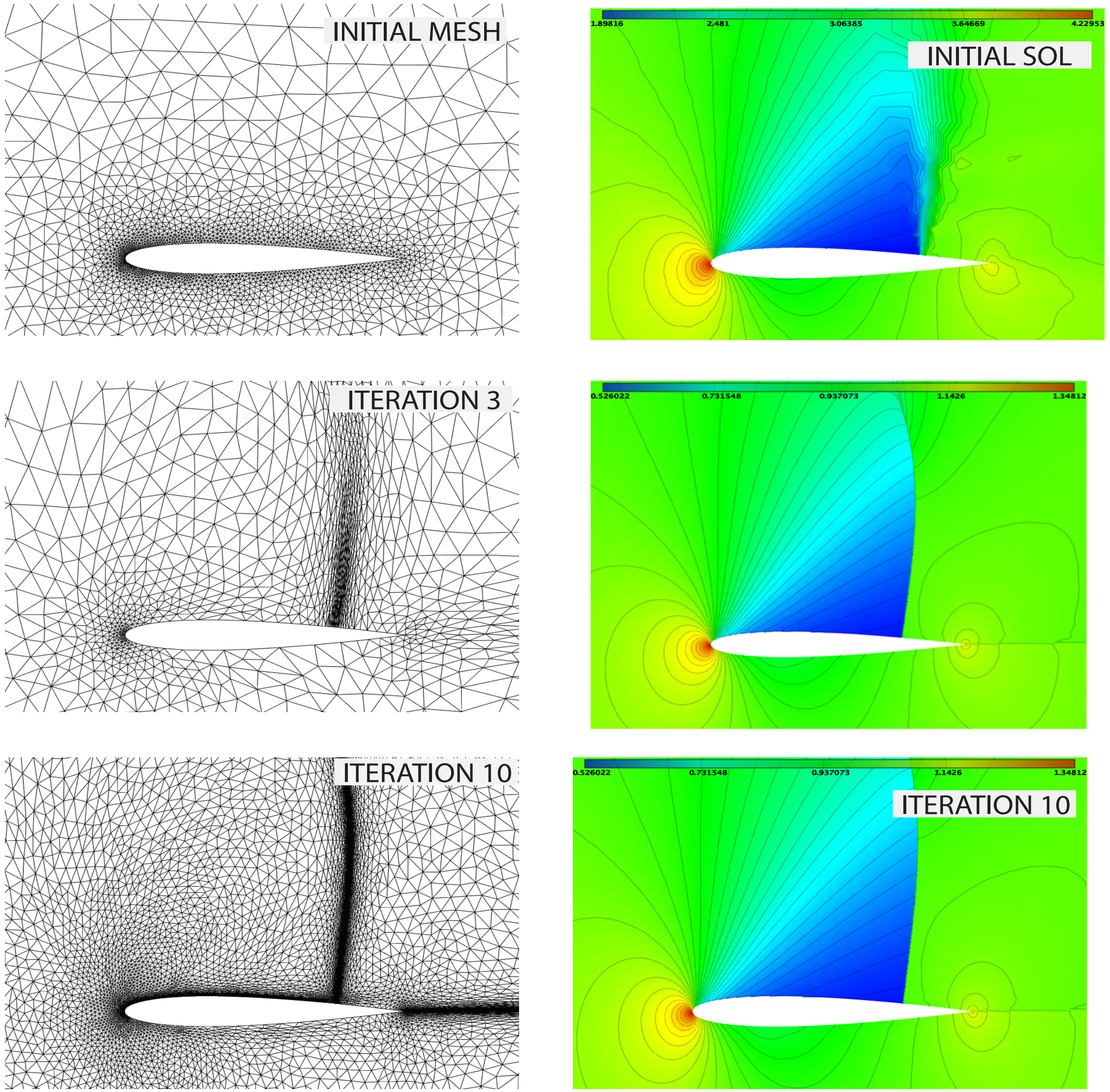


- CPU time : 4 hours
- If done with uniform refinement : 10^{18} tetrahedra
- If done with uniform mesh : 1000 years of computation

- **Mesh adaptation :**
  - Early capturing of physical phenomena
  - Second order convergence on flows with shocks
  - Capture all the scales of the flow

Mesh Adaptation is a non-linear problem





Metric-based framework

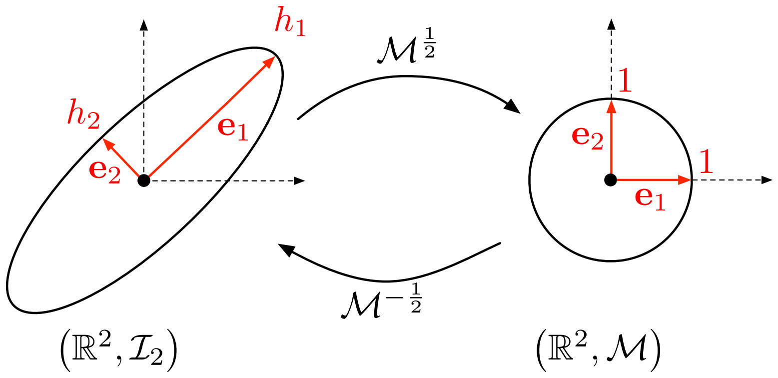
- Main idea : change the distance and volume computation used the mesh generator [George, Hecht and Vallet., Adv. Eng. Software 1991]
- Fundamental concept : The notion of metric Riemannian metric space
  - Euclidean Metric space
  - Riemannian Metric space

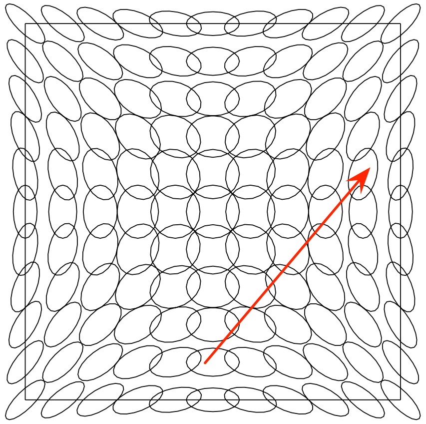
$$\ell_{\mathcal{M}}(\mathbf{ab}) = d_{\mathcal{M}}(\mathbf{a},\mathbf{b}) = \|\mathbf{ab}\|_{\mathcal{M}}$$

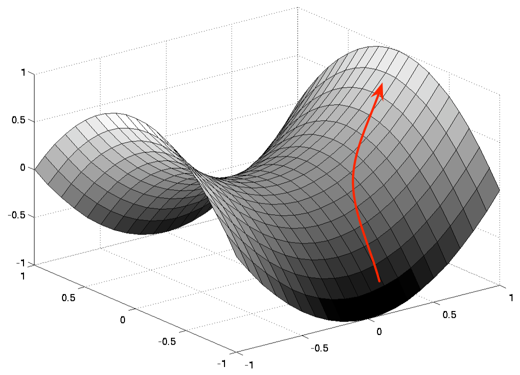
$$\langle \mathbf{u},\mathbf{v}\rangle_{\mathcal{M}} = \langle \mathbf{u}, \mathcal{M}\mathbf{v}\rangle = {}^t\mathbf{u}\mathcal{M}\mathbf{v}$$

$$\mathbf{M} = (\mathcal{M}(\mathbf{x}))_{\mathbf{x}\in\Omega}$$

$$\ell_{\mathcal{M}}(\mathbf{ab}) = \int_0^1 \sqrt{{}^t\mathbf{ab}\,\mathcal{M}(\mathbf{a}+t\,\mathbf{ab})\,\mathbf{ab}}\,\mathrm{d}t$$







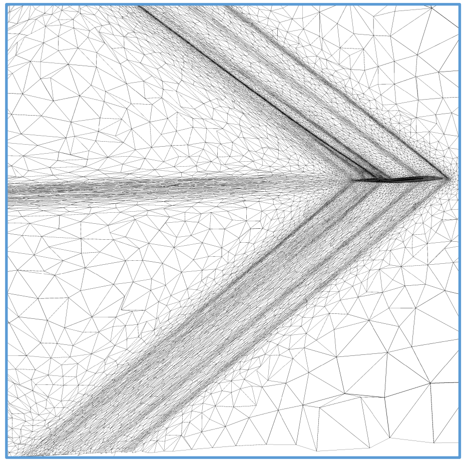
Metric-based error estimates

- Scope : From numerical solution → derive a metric-field to drive adaptivity

[Venditti and Darmofal, JCP 2003], [Jones et al., AIAA 2006], [Power et al., CMA 2006], [Wintzer et al., AIAA 2008], [Leicht and Hartmann, JCP 2010],

Multi-scale error estimates :

- Derive the ’best’ mesh to compute the characteristics of a given solution W
- Optimal control of the interpolation error in L^p norm:



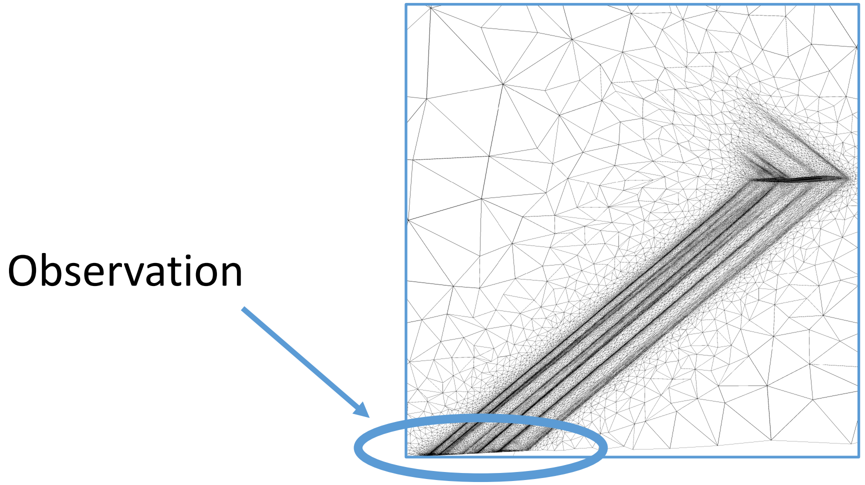
$$\|W - \Pi_h W\|_{L^p(\Omega_h)}$$

Goal-oriented error estimates

- Derive the ’best’ mesh to observe a given functional

$$j(w) = (g, w)$$

- Optimal control of the interpolation error in L^p norm:

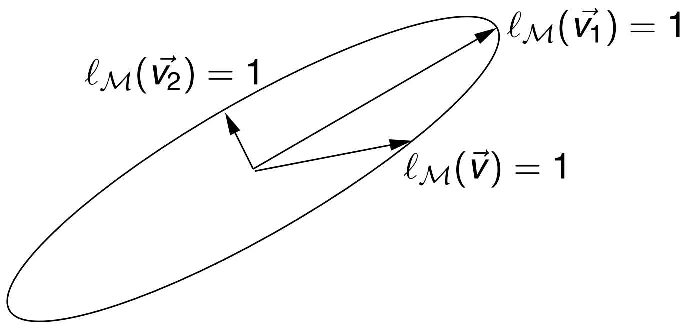


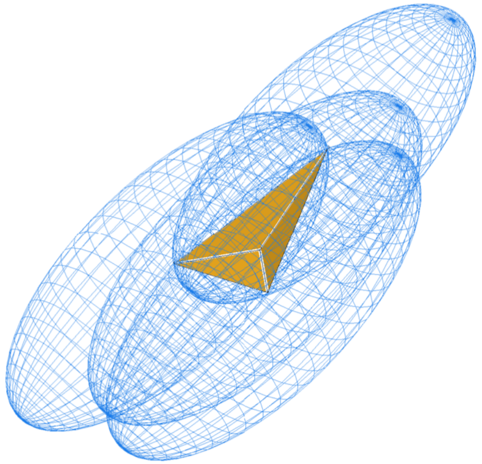
$$|J(W) - J(W_h)| \approx \left\|\nabla W^* \cdot \left(\mathcal{F}(W) - \Pi_h \mathcal{F}(W)\right)\right\|_{L^2(\Omega_h)}$$

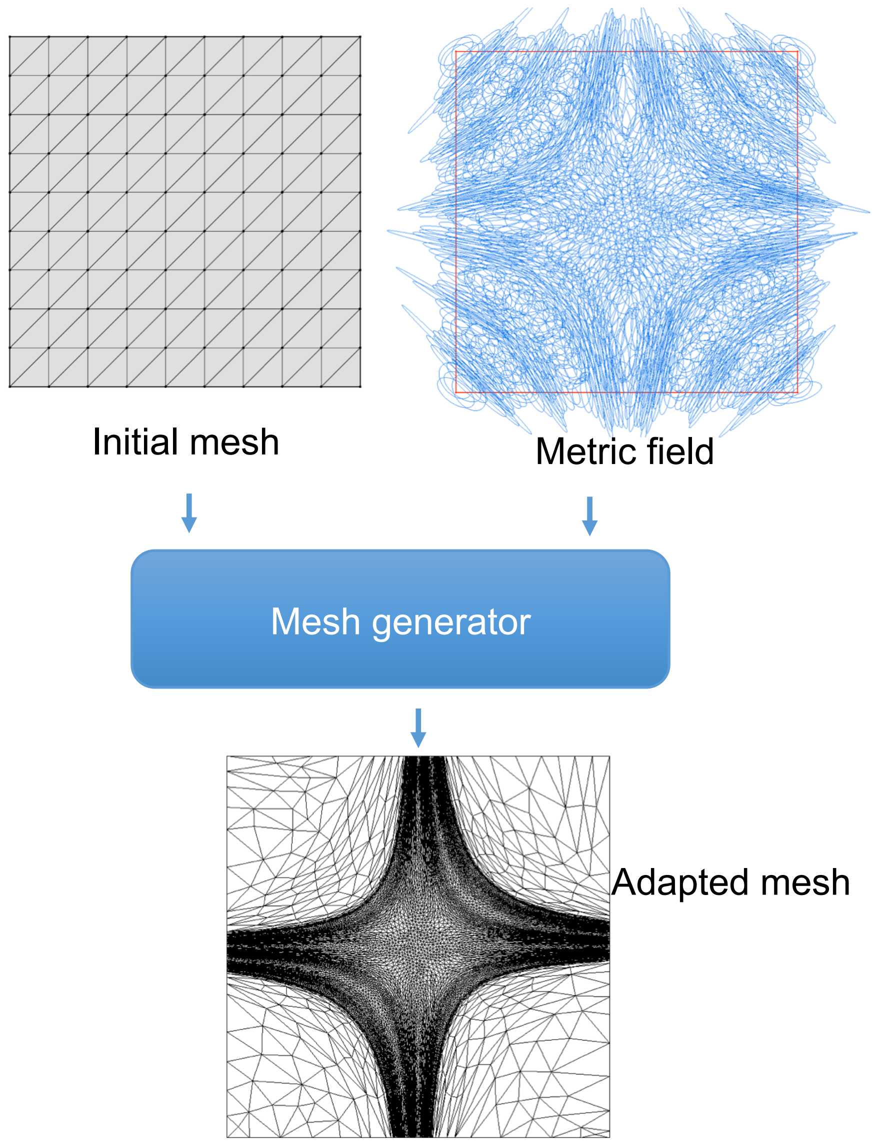
Anisotropic mesh generation

- Generate a unit mesh w.r.t. the input metric field.
- Local mesh operations are performed:
- Standard mesh operators: point insertion/deletion, edge swap/collapse, etc.
- In AMG: all these operations are embedded in

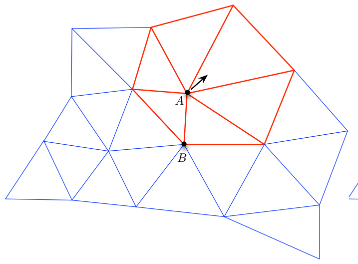
one cavity-based operator



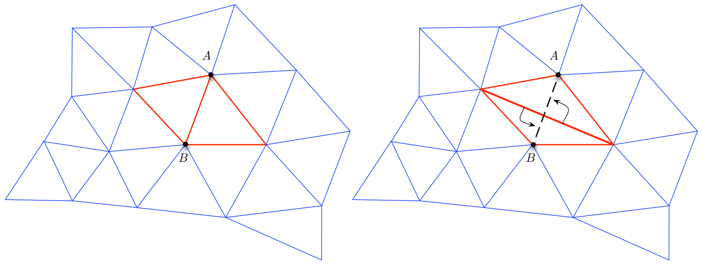




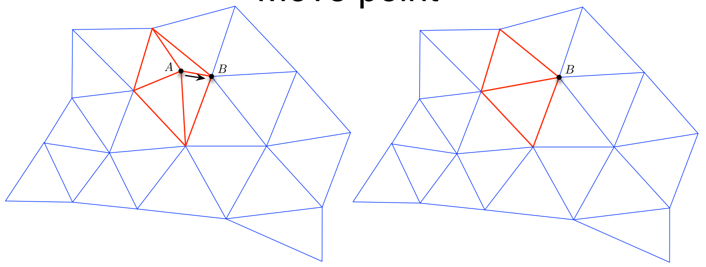
Standard meshing operators



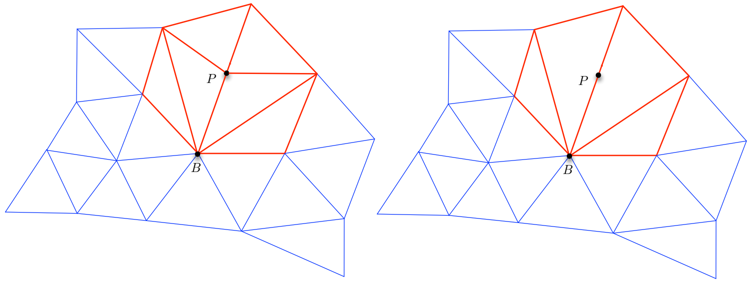
Move point



Swap edge



Collapse edge



Point insertion

- Step 1 : Generate a unit mesh
  - Split long edges in the metric
  - Collapse small edges in the metric
- Step 2 : Optimization
  - Perform point smoothing and swaps

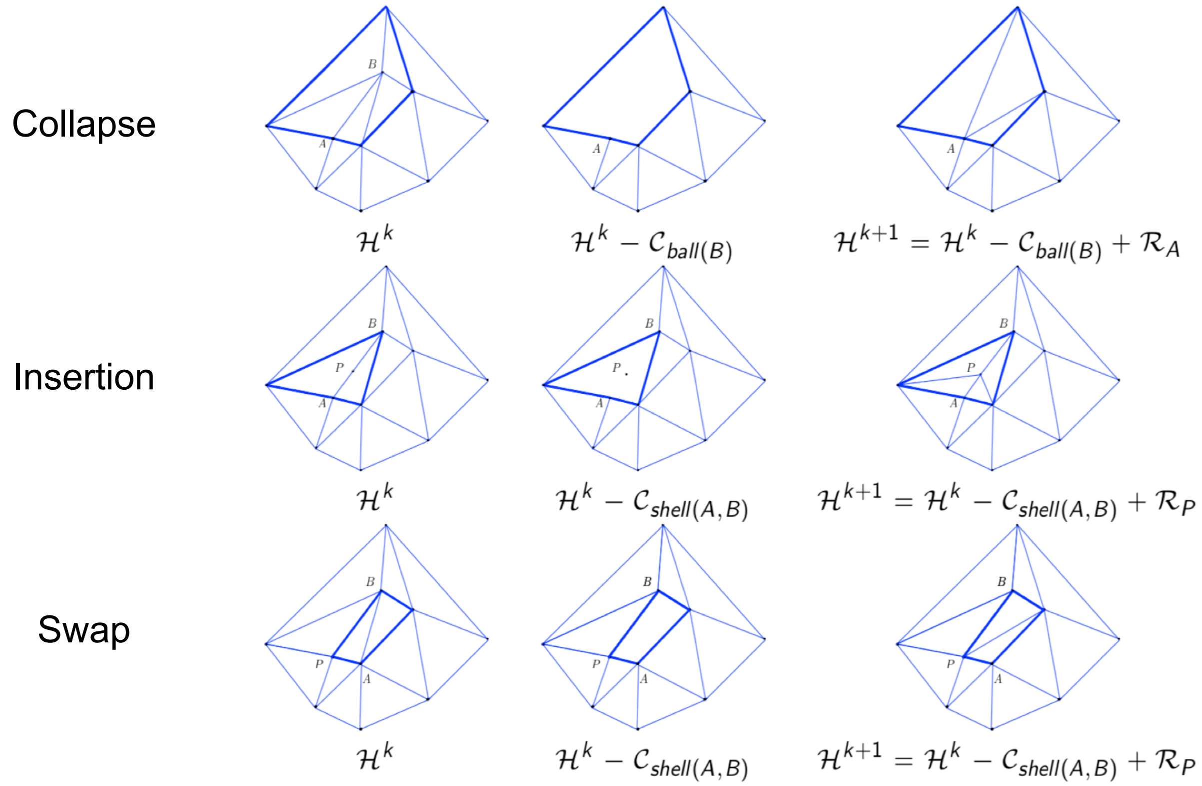
- All these operators are applied sequentially and are monitored by a quality function

Unique cavity-based operator

[Loseille and Löhner, AIAA 2010, Loseille et Menier, IMR 2013]

▸ In PyAMG:

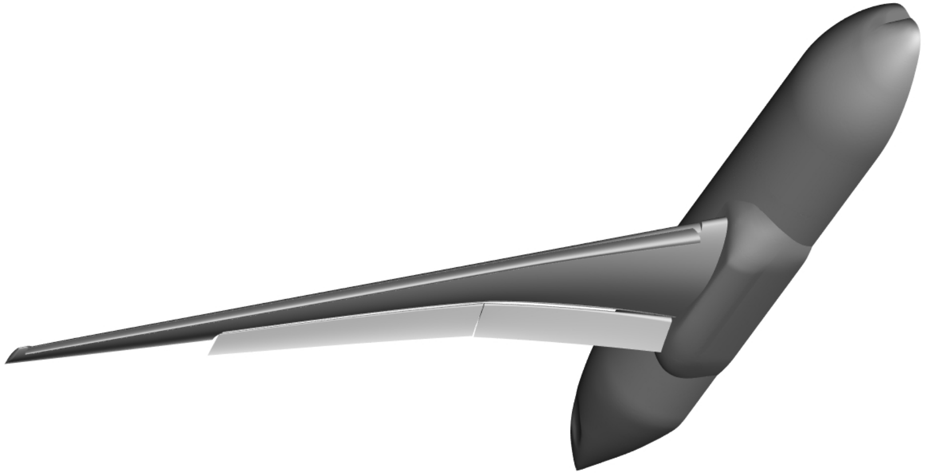
- Each mesh operation corresponds to a node (re)insertion



- Unified : all-in-one
- Generalized : multiple-operators at once
  - Do either line, surface or volume
  - Handle non-manifold geometries
  - Easy to maintain and update
  - Uniform speed for all operators

HL-CRM

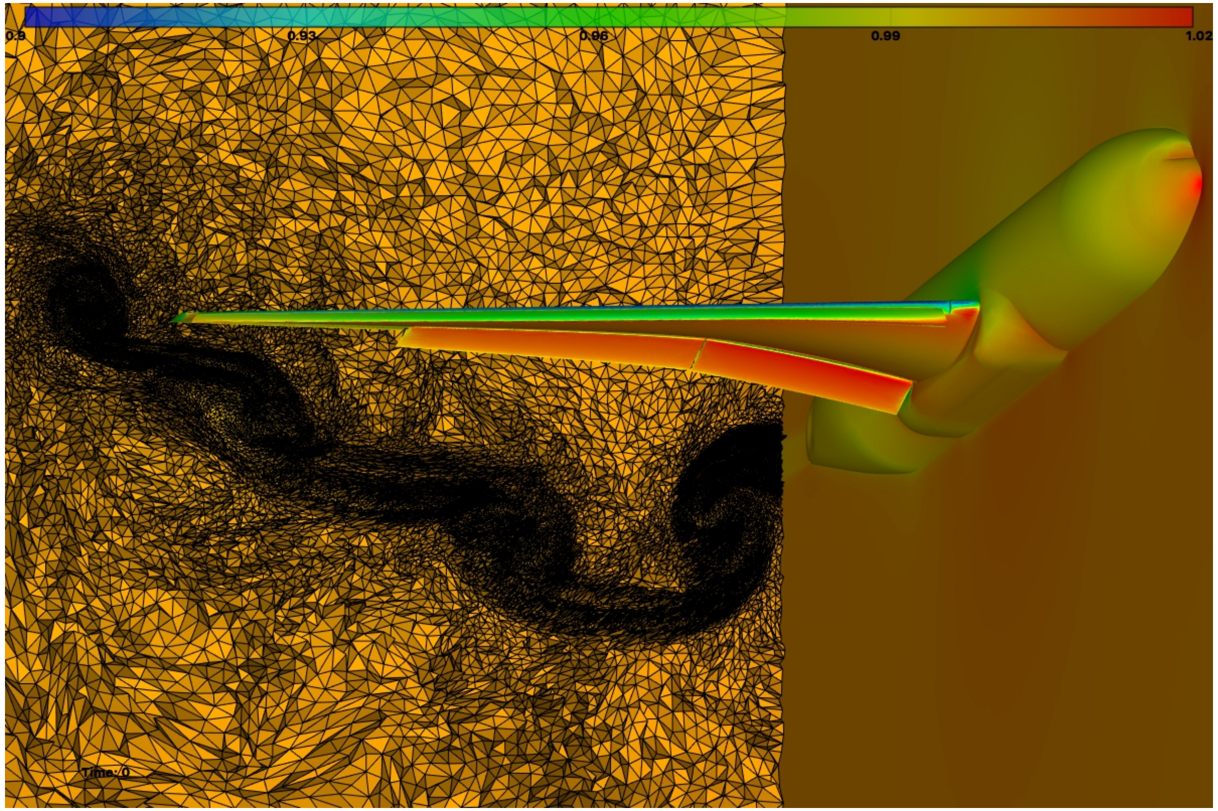
- Mach 0.2, 16 degrees, 3,2 Million
- Meshing process : start from a coarse mesh
- Geometry : P3 curved mesh (built from CAD)

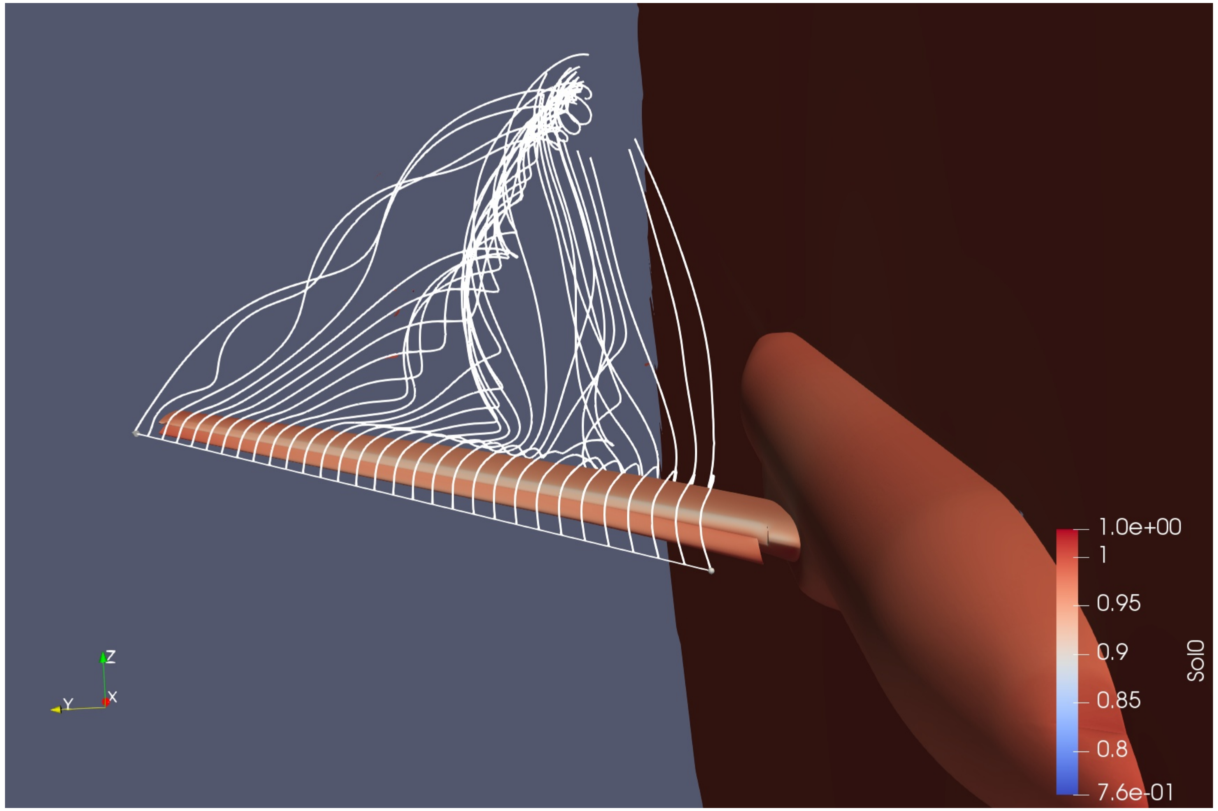


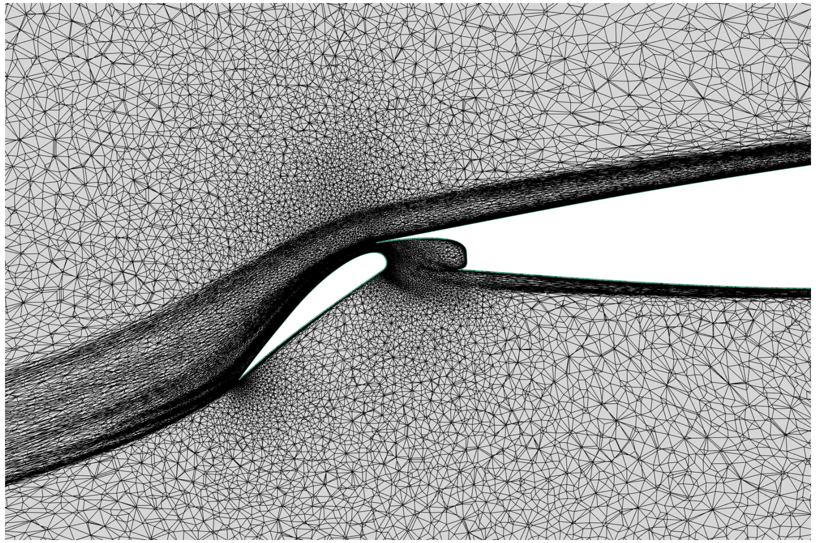
Software packages

- Wolf flow solver : FV/FE, HLLC scheme, Implicit, SA-Neg
- Goal oriented RANS error estimates : lift functional
- Fully anisotropic : no quasi-structured boundary layer mesh inserted

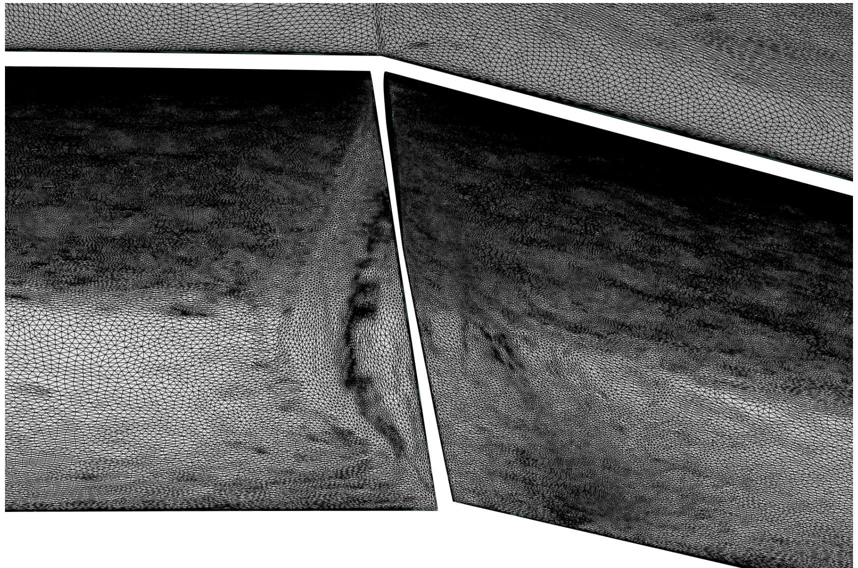
[L. Frazza Phd Thesis, 2018], [Alauzet and Frazza, Eccomas 2018], [Michal et al., AIAA 2018]

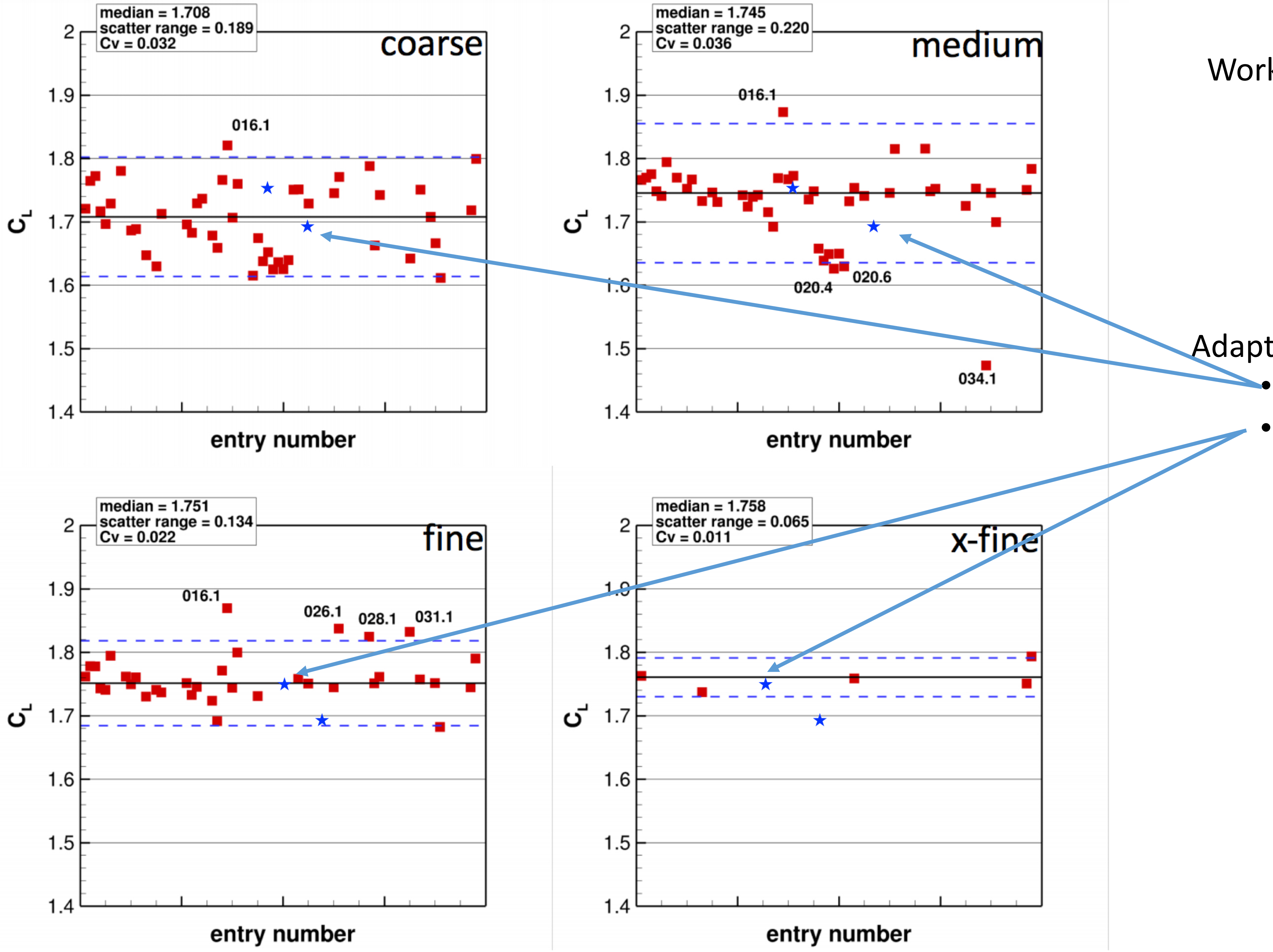






- Final mesh : 13 M vertices
- 10 Adaptations





Workshop grids :

- Coarse : 8 Million vertices
- Medium : 26 Million vertices
- Fine : 70 Million vertices
- X-Fine: 206 Million vertices

Adaptive grids : 10 adaptations

- Level 1 : 3,1 Million vertices
- Level 2 : 13 Million vertices

Outline

- Background on anisotropic mesh adaptation
  - Metric-based mesh generation
  - Error estimates
  - Unique cavity-based operators
  - Numerical examples : RANS Adaptation
- PyAMG interface and SU2
  - Python bindings
  - Numerical adaptive examples

Challenges of exposing adaptivity

- Automaticity : Minimal user intervention/parameters
- Geometry : surface approximation/projection/CAD
- Mesh Quality/Anisotropy ⟷ Solver Stability/Accuracy
- Error estimates : theoretical developments are still on-going

Typical issues with CFD solvers

- Associativity of boundary conditions to initial Geometry/CAD are lost ..
- Restart capabilities
- Strong solvers : stability and convergence on highly anisotropic elements

PyAMG : technical details

- Use of Python : Mesh, Solution and Options are dictionaries

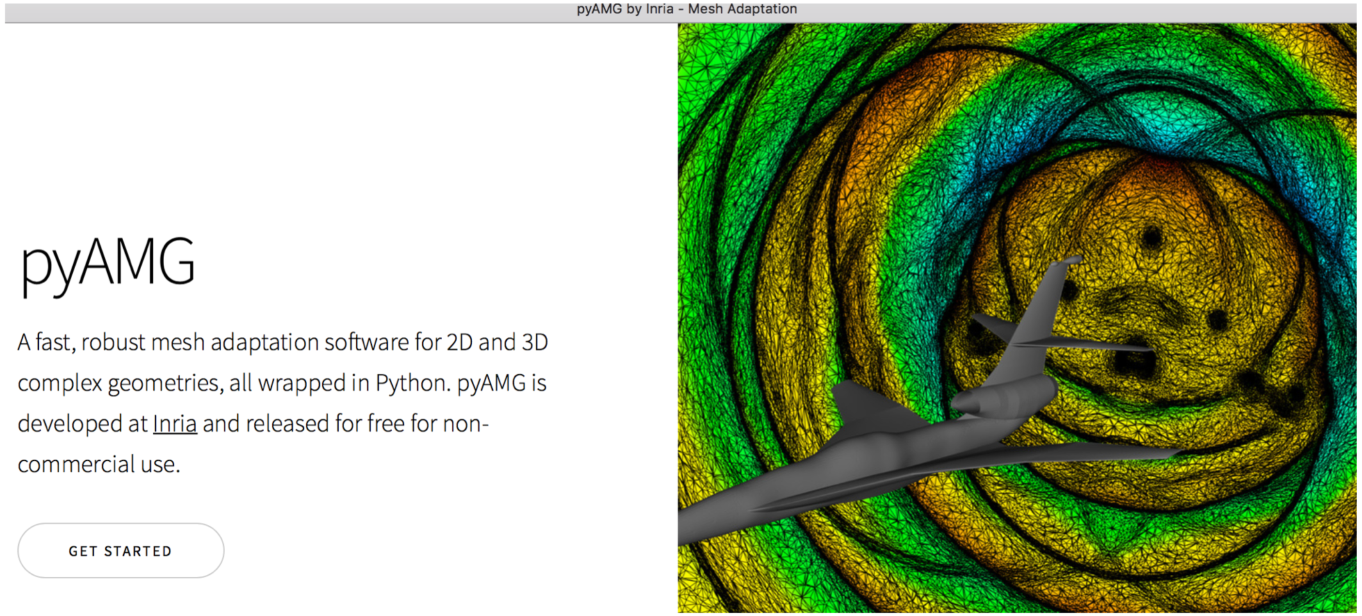
- Expose first a reduced set of capabilities
  - P3 High-order geometry — *prior to expose* CAD projection
  - Multi-scale error estimate — *prior to expose* Goal-oriented
  - Fully anisotropic — *prior to expose* adaptive quasi-structured BL

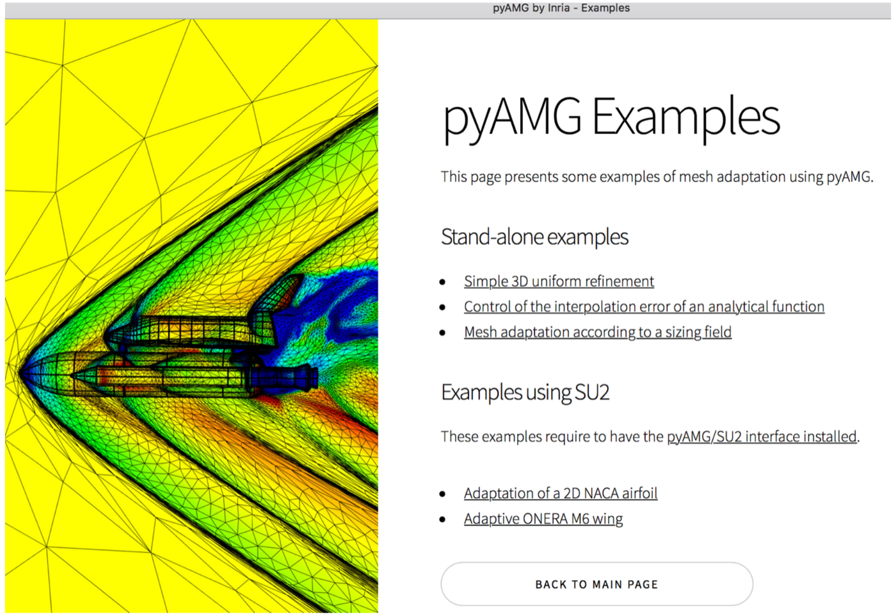
- Have a reduced number of user-parameters for mesh adaptation
  - Accuracy level
  - Gradation of the mesh
  - Sensor

- Full integration with SU2 in config file

PyAMG website

- https://pyamg.saclay.inria.fr





PyAMG : example of mesh definition

```
import pyamg
mesh = {}
mesh['xy']          = [ [0,0], [1,0], [1,1], [0,1] ]
mesh['Triangles'] = [ [1,2,3,0],[3,4,1,0]]
print "Writing initial_mesh.mesh"
pyamg.write_mesh(mesh, "initial_mesh.mesh")
remesh_options = {'hmax':0.5}
print "Refining mesh and writing refined_mesh.mesh"
adapted_mesh = pyamg.adapt_mesh(mesh,remesh_options)
pyamg.write_mesh(adapted_mesh, "refined_mesh.mesh")
```

- Mesh-related Keywords include : ‘xy’, ‘xyz’, ‘Triangles’, ‘Tetrahedra’, ’Edges’, ’Corners’
- Solutions is stored in key ‘sensor’
- Option-related keywords include : ’hmax’, ’hmin’, ‘Lp’, ‘gradation’

PyAMG : Example of mesh adaptation

```
# Define remeshing options
remesh_options              = {}
remesh_options['Lp']        = 2
remesh_options['gradation'] = 1.5
remesh_options['target']    = 20000

# Adapt the mesh, perform 5 iterations
print "   Anisotropic adaptation : iteration 1"
msh3d_aniso = pyamg.adapt_mesh(msh3d, remesh_options)

for ite in range(2,5):
  print "   Anisotropic adaptation : iteration %d " %(ite)
  msh3d_aniso['sensor'] = create_sensor(msh3d_aniso)
  msh3d_aniso           = pyamg.adapt_mesh(msh3d_aniso, remesh_options)

# Output final mesh
pyamg.write_mesh(msh3d_aniso,"cube_aniso.meshb")
```

PyAMG-SU2 : Example

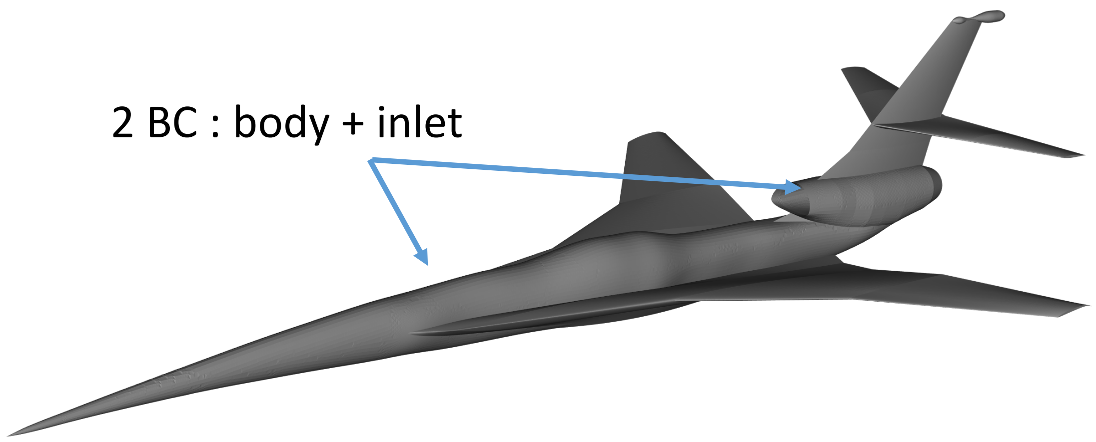
```
% ------------- MESH ADAPTATION PARAMETER ------------%

% Mesh size parameters
ADAP_SIZES= (2000, 4000, 8000)

% Number of iterative loops performed for each prescribed size
ADAP_SUBITE= (2, 2, 2)

% Number of CFD iterations for each mesh size
ADAP_EXT_ITER= (1000, 1000, 1000)

% Prescribed residual reduction for each mesh size
ADAP_RESIDUAL_REDUCTION= (3, 3, 3)

% Sensor used for mesh adaptation (MACH, PRES, or MACH_PRES)
ADAP_SENSOR=  MACH

% Max and min edge sizes
ADAP_HMAX=  200
ADAP_HMIN=  1e-8

% Prescribed mesh gradation
ADAP_HGRAD= 1.3

% Output adapted mesh
MESH_OUT_FILENAME= M6_adap.su2

% Final adapted restart solution
RESTART_FLOW_FILENAME= M6_adap.dat
```

- Everything is contained in the config file
- PyAMG bindings are transparent to the user

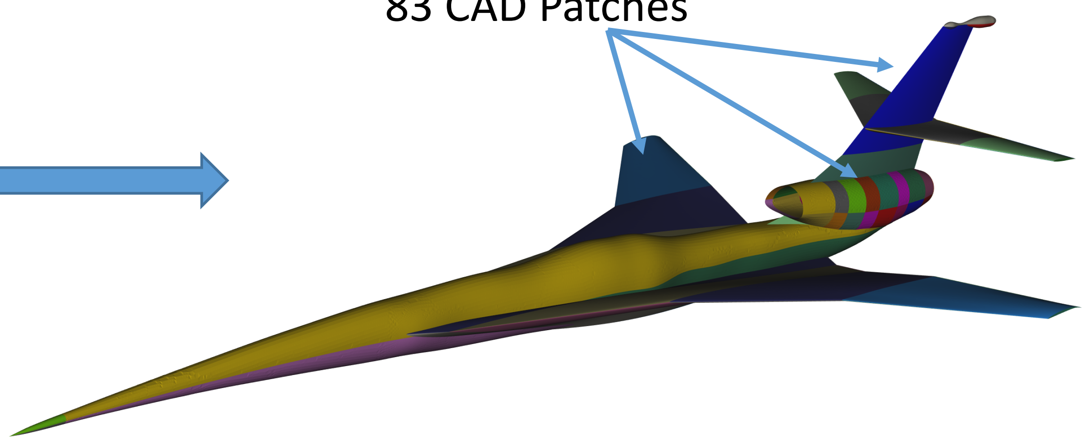
```
$ python mesh_adaptation_amg.py -f adap_ONERAM6.cfg
```

PyAMG-SU2 : Behind the scene

- Convert SU2 data structures into Python dictionaries
- Expand/Split MARKERS to be as close as possible of the most detailed geometry description (CAD Patch)



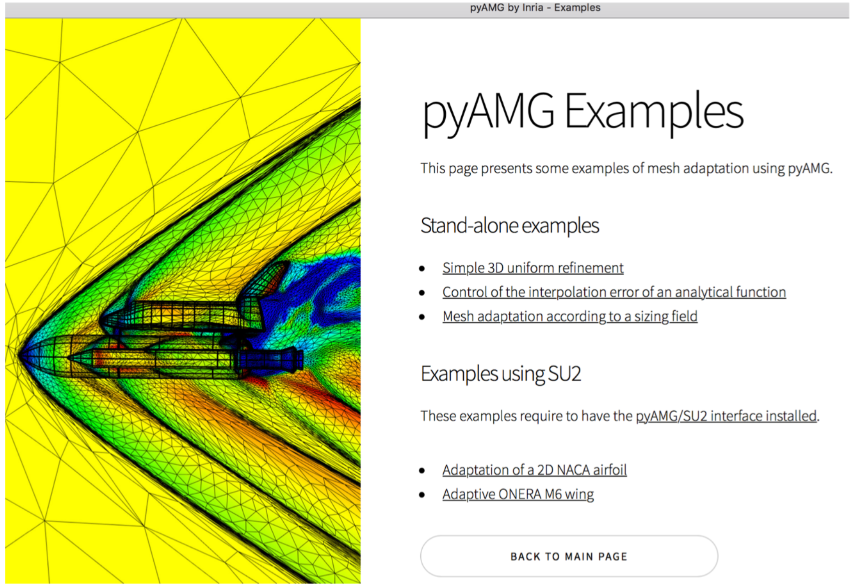
CFD Boundary condition



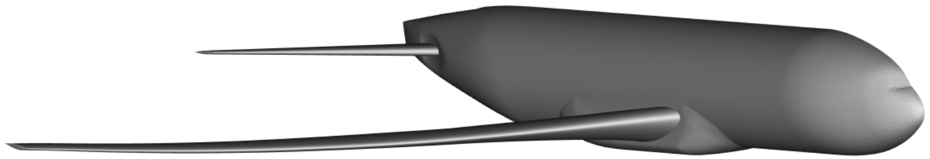
Good practice

PyAMG-SU2 : current status

- Inviscid and RANS with frozen boundary layers are support
- Support to fully unstructured boundary layers is on-going
- The adaptive script is still a custom branch of SU2



PyAMG-SU2 : Inviscid CRM

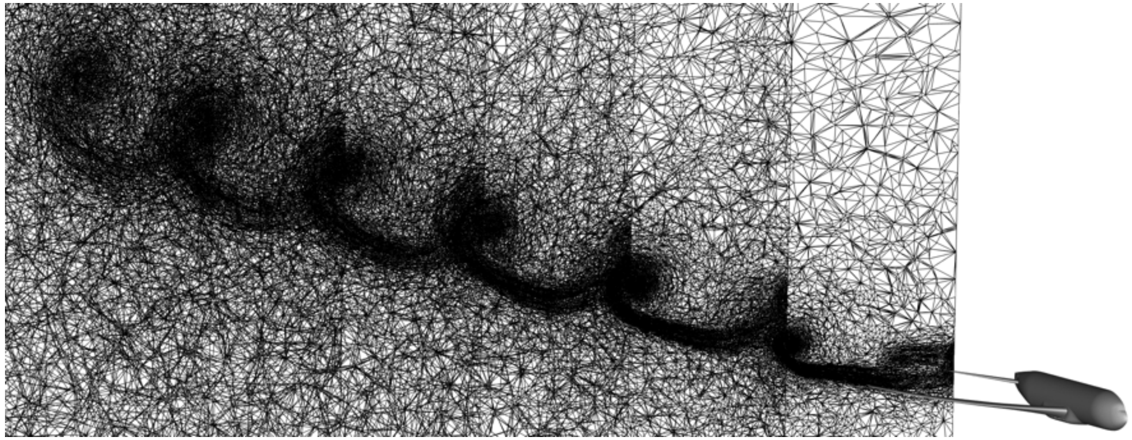


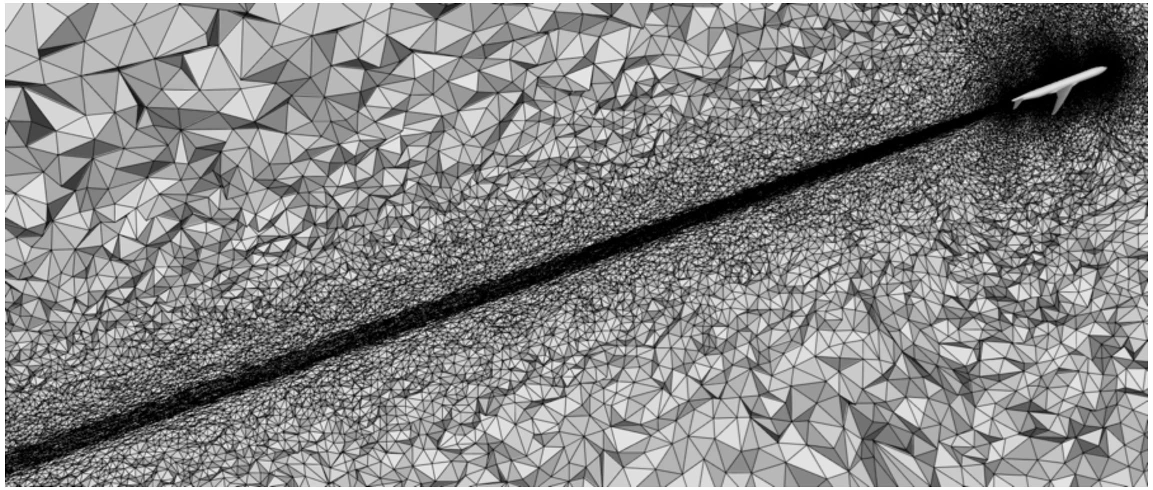
- Mach 0.85, 2.133 degrees, Inviscid
- Meshing process : start from a coarse mesh
- Geometry : P3 curved mesh (built from CAD)

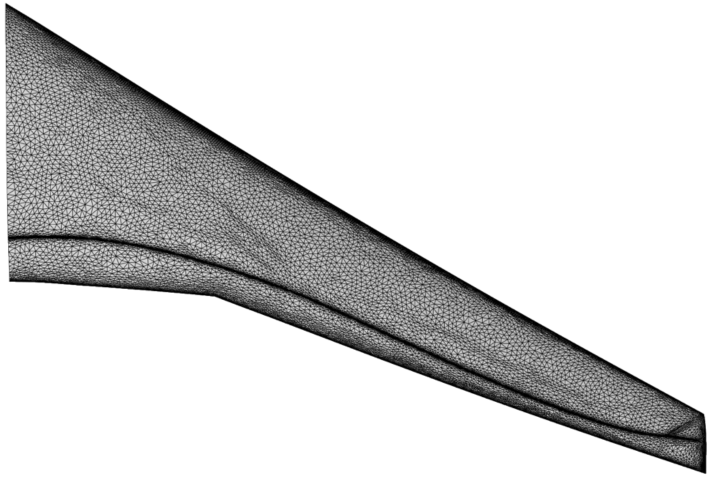
Software packages

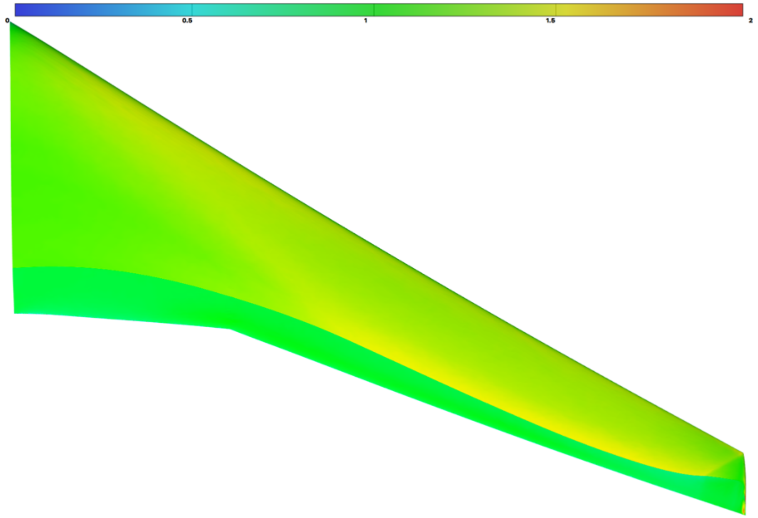
- SU2 : JST, MG, FMGRES
- Multi-scale error estimate : Mach Number
- Final mesh 3 019 832 vertices

PyAMG-SU2: Inviscid CRM

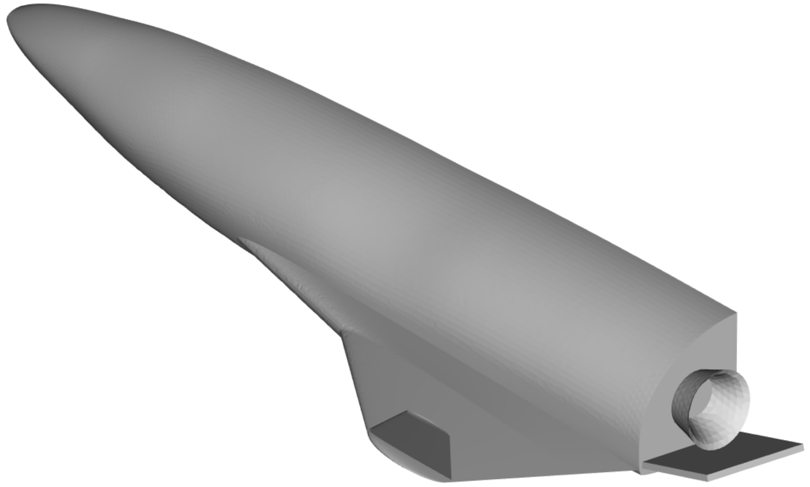








PyAMG-SU2 : RANS Case

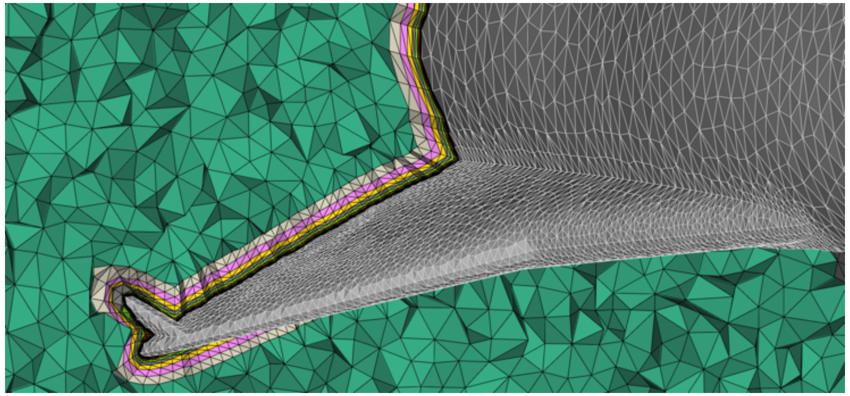


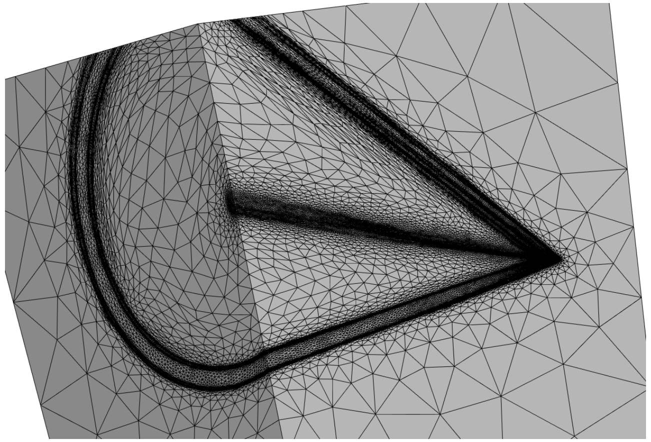
- Mach 2.0, 10 degrees, 1.65 Million
- Meshing process : start from a coarse mesh
- Geometry : P3 curved mesh (built from CAD)

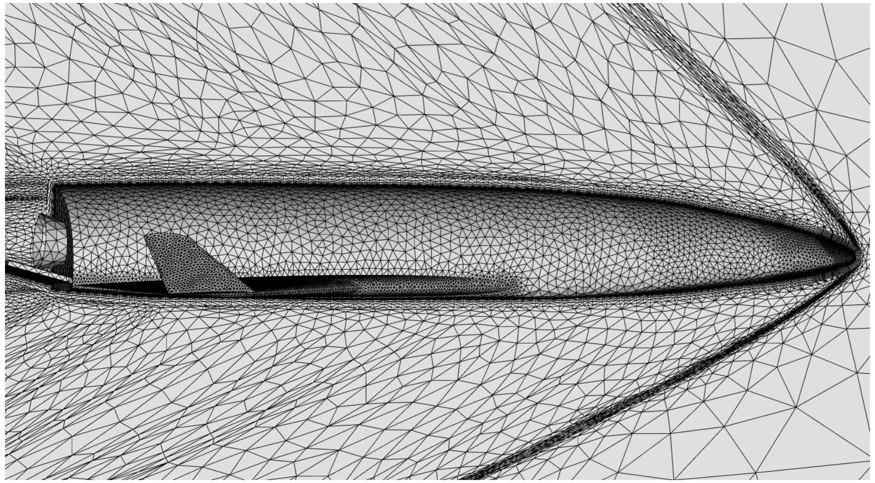
Software packages

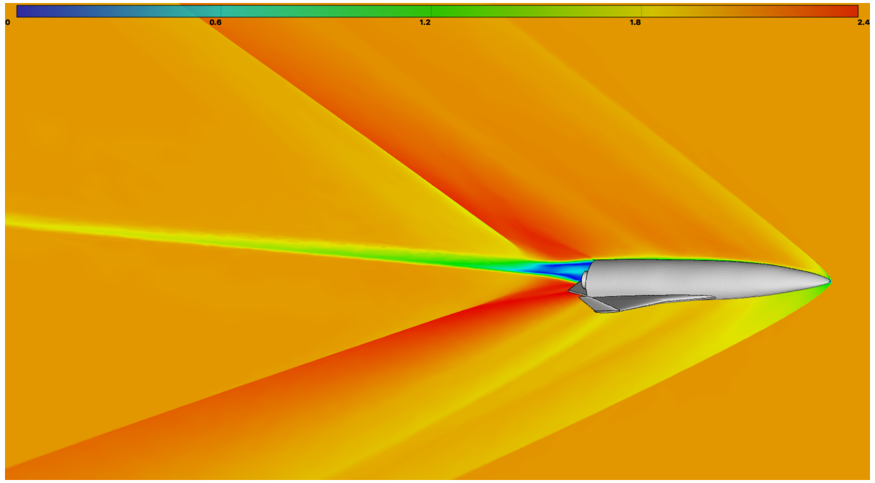
- SU2 : SA, JST, FMGRES
- Multi-scale error estimate : Mach Number
- Frozen boundary layer
- Final mesh 1 343 684 vertices (outside BL)

PyAMG/SU2: RANS case









Conclusion

- Beta version of adaptive mesh generation in 2D/3D
- Error estimates, interpolation, surface, volume mesh generations
- Inviscid runs, RANS adaptations

On-going work

- Increase the set of documented adaptive test cases in 2D/3D
- Detail best practice to start a new adaptive computation (Geometry)

Stay updated

- Visit pyamg.saclay.inria.fr for more details
- Mailing list : send an email to pyamg-request@inria.fr with subscribe as topic

Acknowledgements

- Frédéric Alauzet and Loïc Frazza :

  Wolf flow solver, Adjoint RANS error estimates
- Julien Vanharen and Mathieu Rigal :

  Python bindings (PyMeshb, PyViz)
- Unstructured Grid Adaptation Working Group

  https://github.com/UGAWG
- Visualization software : ViZiR team (INRIA)

  https://vizir.inria.fr/

Thank you for your attention !

pyamg.saclay.inria.fr

pyAMG

A fast, robust mesh adaptation software for 2D and 3D complex geometries, all wrapped in Python. pyAMG is developed at Inria and released for free for non-commercial use.

GET STARTED