AIAA SciTech Forum
11–15 & 19–21 January 2021, VIRTUAL EVENT
AIAA Scitech 2021 Forum

10.2514/6.2021-0476

Towards Adaptive Surface Meshing in Strand Grid Framework

Vinod Lakshminarayan[1],
Jayanarayanan Sitaraman[2], and Andrew Wissink[3]

[1] *Science & Technology Corporation, NASA Ames Research Center, Moffett Field, CA*
[2] *Parallel Geometric Algorithms LLC, Sunnyvale, CA*
[3] *U.S. Army Combat Capabilities Development Command Aviation & Missile Center, Moffett Field, CA*

The strand grid approach is a flow solution method where a prismatic-like grid using "strands" is grown to a short distance from the body surface to capture the viscous boundary layer and the rest of the domain is covered using an adaptive Cartesian grid. The approach offers several advantages in terms of nearly automatic grid generation and adaptation, ability to implement fast and efficient flow solvers that use structured data in both the strand and Cartesian grids, and the development of an efficient and highly scalable domain connectivity algorithm. Earlier works by the authors presented meshing and domain connectivity algorithms to automatically generate meshes for geometrically and topologically complex test cases; and introduced a fully parallel and highly efficient strand grid solver called `mStrand`. An initial attempt has also been made in automating the generation of surface meshes starting from a closed parameterized surface. This paper aims at developing an infrastructure in the strand grid framework to support fully automated surface mesh adaptation, which will allow better use of the available degrees of freedom and improve the overall automation process. The developed infrastructure is verified by performing uniform surface mesh refinement on the ROBIN fuselage. In addition, an initial demonstration of a solution-based surface mesh refinement is provided for the ONERA M6 wing configuration.

I. Introduction

The use of computational fluid dynamics (CFD) arise in many scientific and engineering applications. One of the most notable difficultly in applying CFD early in the design process is the lack of automation in mesh generation. With the advent of High Performance Computing (HPC), the fraction of time required for mesh generation and problem setup has increased significantly. Lack of automation not only increases the time required to set up and perform the analysis, it also introduces an immeasurable source of variability. That is, the quality of the simulation results depends on the quality of the grids used, and an engineer with a wealth of skills and experience in grid generation techniques will typically generate higher quality grids, which results in higher quality results. Only through automation can reproducible results be generated by quality rotorcraft engineers that are not highly trained in CFD grid generation procedures.

Adaptive Cartesian cut-cell[1–4] and immersed boundary [5–7] methods achieve a high degree of automation in grid generation. Using adaptive mesh refinement (AMR) they can automatically generate Cartesian representations around almost any closed geometry, regardless of geometric complexity. As a result these methods are used regularly today in design trade off analysis tools. The shortcoming of these methods is their inability to resolve thin turbulent boundary layers in the high Reynolds number flows typically encountered in full-scale aerospace vehicles. For a typical rotorcraft fuselage the viscous skin-friction drag can be as much as half the total drag. Inability to resolve the viscous turbulent boundary layer is therefore a significant hindrance.

Extensions of the cut-cell method to resolve the boundary layer have been explored by Berger and Aftosmis[8,9] using a subgrid wall model. Brehm et al[10] have explored extensions of immersed boundary methods using wall modeling. While these novel methods offer a potential alternative to traditional body-fitted grid generation techniques and certainly deserve further exploration, they are still in the early development

stages and have not demonstrated applicability for three dimensional full-scale rotorcraft and fixed wing configurations.

Traditionally, there are two primary types of body-fitted gridding strategies - structured and unstructured grids. Flow solution on a structured grid is efficient and allow use of higher order algorithms. However, generating a structured grid, with either multi-block structured and/or overset meshes even for mildly complex geometries can be extremely challenging and demands significant user expertise. On the other hand, unstructured grids offer more flexibility in grid generation and problem setup for geometrically-complex bodies such as fuselages and hubs. However, automating the unstructured grid generation is still not very straightforward. Further, the overall computational cost to provide a level of accuracy comparable to a structured grid solver is significantly higher for the unstructured grid solvers. Therefore, in order to perform high-fidelity modeling and simulation in a nearly automated manner, an approach is needed that combines high efficiency comparable to structured grid solvers with a more automated mesh generation strategy.

An alternative strategy to the pure Cartesian-based approaches for automated gridding is the overset strand/Cartesian framework (referred to as strand grid framework), introduced by Meakin et al.[11] in 2007. In this approach, a thin body-fitted "strand" mesh is constructed with high-aspect-ratio cells at the wall in order to resolve the viscous turbulent boundary layer. The strand mesh generation process is fully automated by extending a viscous-quality prismatic mesh directly from a surface tessellation composed of triangles and quadrilaterals. A near-body volume grid is constructed by inflating the surface grid along curves that can be represented with just a few parametric quantities. These curves, referred to as "strands", are typically straight lines represented by a pointing vector and a length (see Fig. 1). Strands extrude a short distance from the solid boundary and intersect with adaptive Cartesian grids, which cover the rest of the domain to the outer boundaries. The strand and the Cartesian grids are connected through an overset interface. The Cartesian grid is automatically generated based on the wake spacing, which is the normal spacing at the outer boundary of the strand grid.

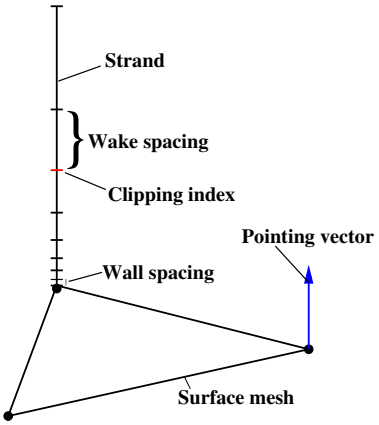


Figure 1. Strand pointing vector and clipping index.

Strand near-body meshes combined with Cartesian off-body grids not only have the potential for automatic viscous mesh generation and adaptation but can also provide other advantages. First, both strand and Cartesian meshes possess some grid structure, facilitating efficient implementations of high-order accurate discretizations and solution methods such as high-order finite differencing, line-implicit solvers, and directional multi-grid coarsening. Second, each strand can be represented with minimal information such as strand direction and length, thus reducing the entire volume mesh description to primarily a surface definition. This allows use of a highly efficient and scalable domain connectivity package[12] to facilitate data exchange between the two mesh types.

In the past, current authors have worked on developing specialized strand solver called `mStrand`[13,14] as a part of HPCMP CREATE^{TM}-AV Helios framework, which provides a high-fidelity analysis capability to the Department of Defense (DoD) for the acquisition of new rotary-wing aircraft. The authors have demonstrated that the new strand solver is efficient and highly scalable, with accuracy similar to other legacy flow solvers. More recently, current authors[15] made advancements to strand volume meshing to handle topologically and geometrically complex test cases with some limitations. From empirical evaluation, it became evident that it is nearly impossible to prevent self-intersections in strand grids for realistic geometries, especially if the strand grids are to extend a reasonable distance from the wall. This earlier work[15] also developed capabilities in the domain connectivity methodology for the treatment of self-intersecting meshes.

Application and verification of the strand grid framework to accurately predict flow-field around complex geometries using volume meshes that are generated automatically in parallel at run time was presented in Ref. 16. The results show that the solutions obtained with the strand framework using automatically generated volume meshes are comparable to well-established flow solution procedures for most cases. In certain cases, the solution accuracy were only satisfactory due to errors arising from less than desirable mesh quality. To rectify these problems, the meshing algorithms were further advanced in Ref. 17, where

improvements to the mesh smoothing strategy in addition to an algorithm to natively generate multi-stranded mesh were presented. A more recent work[18] also made an initial attempt towards automating the surface mesh generation to simulate simple fuselages and rotor blades.

The current work aims at developing an infrastructure in the strand grid framework that can support fully automated surface mesh adaptation, to allow better use of the available degrees of freedom, to either improve the solution accuracy provided a given amount of computational resource or reduce the computational cost required to attain a desired accuracy. The use of solution-based mesh adaptation in the near-body grid also improves the overall automation process by removing the need for any human knowledge about the flow solution. This paper describes the first steps in a framework to enable automated surface refinement for strand meshes over the course of a simulation. The framework can currently handle runtime generation of a new mesh, interpolation of solution from the old mesh to the new mesh and application of any underlying mesh motion/deformation. The framework ensures that the solution procedure continues seamlessly on the new mesh after adaptation without any human intervention. The developed framework is verified by performing uniform surface mesh refinement on the ROBIN fuselage. In addition, an initial demonstration of a solution-based surface mesh refinement is provided for the ONERA M6 wing configuration.

The remainder of this paper is organized as follows. Section II provides a brief description of the flow solvers that are part of the strand grid framework in Helios. Section III describes the automated surface meshing capability. Description of the automated strand/Cartesian volume mesh generation strategies is given in section IV. A brief introduction to the newly developed infrastructure for handling adaptive meshes is provided in section V. Next, the results are presented in section VI. Finally, concluding remarks are offered in section VII.

II. Description of Flow Solvers

This section provides a brief description of the near-body strand solver, mStrand,[13] and the off-body Cartesian solver, SAMCart.[19]

II.A. Strand Near-Body Solver (mStrand)

`mStrand` is a specialized strand grid solver that uses a vertex-centered finite-volume spatial discretization. The Reynolds-averaged Navier-Stokes (RANS) equations in a general moving coordinate system in three dimensions is solved. `mStrand` accommodates both quadrilateral and triangular surface elements and handles general prismatic meshes in the normal (strand) direction. The Roe's approximate Riemann solver[20] is used to compute the inviscid conservative fluxes. The second-order gradients are limited using a differentiable form of Van-Albada's limiter[21] in both the streamwise and the strand directions. The solver incorporates a second-order implementation of full Navier-Stokes for the viscous terms (no thin-layer approximation) and the option of using the Spalart-Allmaras[22] or the Menter's $k-\omega$ SST[23,24] turbulence models implemented with a first order discretization. More recently, the solver is also equipped with a first order discretization of several transition models including the Langtry-Menter,[25] Medida-Baeder[26] and Coder's AFT[27,28] models. Implicit solution is performed using a fully coupled (mean-flow and turbulence equations) preconditioned GMRES.[29] A specialized strand-based preconditioner is constructed from first-order Jacobian terms that require only nearest neighbor contributions. A fully parallel implementation of `mStrand` is obtained by partitioning the surface mesh into contiguous blocks on the basis of surface elements using Metis.[30] More details of the `mStrand` solver can be found in Ref. 13.

II.B. Cartesian Off-body Solver (SAMCart)

A structured adaptive solver SAMCart is used for the Cartesian off-body grid. The parallel mesh adaptive capability is provided by the SAMRAI[31] library and the solution in each block is obtained using a solver called Cart. The Cart solver uses a high-order central differencing scheme - 6th order with 5th order dissipation for the inviscid terms and 4th order for the viscous terms. Cart implements the Spalart-Allmaras[22] and Menter's $k-\omega$ SST[23,24] turbulence models in RANS or DES form. Cart also supports various transition models available in the strand solver. The solver includes an implicit second order BDF2 time integration scheme with LU-SGS and diagonalized ADI implicit operators. Further description of SAMCart can be found in Refs. 32 and 33.

III. Surface Mesh Generation

The starting point for automated meshing is a closed parameterized surface. The ideal surface definition for meshing differs depending on whether the geometry is a bluff body or an aerodynamic body. If the surface is a bluff body, such as a fuselage or hub, the geometry is described through a closed-surface CAD. If the surface has an aerodynamic shape, such as a helicopter rotor or wing, the geometry is described through a selection of 2D airfoil cross-sections, twist, taper, sweep, anhedral, etc.; parameters typically used by rotor design codes. Below are the details of both these surface-meshing approaches.

III.A. CAD-based Surface

To support surface mesh generation from CAD at runtime, HPCMP CREATE^{TM}-MG Capstone v10.1[34] SDK is interfaced with Helios through a python interface. The geometry may be described by a closed-surface CAD format like IGES or STEP. Capstone generates a triangulated surface mesh based on the three user-inputs listed below.

1. **Surface spacing**: Maximum allowed surface spacing. Defaults to 0.01×model length.
2. **Min/max ratio**: Computes the minimum surface spacing from the maximum allowed surface spacing. This parameter allows curvature-based sizing to the surface mesh generation. Defaults to 1.0.
3. **Surface scaling**: Used to scale the mesh to support grid refinement studies. Mesh size varies inversely to the square of this parameter. Defaults to 1.0.

Earlier versions of Helios would generate surface meshes of roughly uniform spacing and required only the surface spacing as input. However, accurate simulation of complex geometries using uniform-spaced meshes can only be achieved with extremely large meshes that has very fine spacing over the entire surface. In the latest release of Helios and as a part of current work, fully-automated simulation of complex geometries is made feasible by tapping into Capstone's ability to perform curvature-based refinement. The curvature-based sizing is determined by the input, min/max ratio.

As a note, since IGES or STEP formats are not guaranteed to be closed, the recommended practice is to open up the CAD in Capstone, verify it is closed, and save it as a `*.cre` CREATE geometry format. Helios can take as input IGES, STEP, or CRE format. However, the case will stop immediately if the code experiences an non-closed geometry which is why the CRE format is recommended.

III.B. Aerodynamic Surface

Rotor designers tend to define blades based on airfoil series cross sections with planform twist, chord, sweep, and anhedral properties. While it is certainly possible to construct a closed CAD geometry from this description, a better approach is to construct the blade surface meshing directly using the information in the blade's original description.

An earlier work by the current authors[18] developed a utility called `bladeGen` for automatic surface meshing of blades in strand grid framework. The tool generates structured quadrilateral-based surface grid through the span of the blade and closes the root and tip caps with a triangulated surface to form a closed a geometry. The `bladeGen` utility interacts with the Helios and the strand grid framework through a python interface; which allows runtime generation of surface grid using the `bladeGen` input file given by the extension ".bgen". All the sectional properties are linearly interpolated between the defined radial stations. The root and tip caps are triangulated using a 2-D constrained Delaunay tessellation.[35] More details about `bladeGen` can be found in Ref. 18.

The current work adds support for creating rounded root and tip caps along with the earlier option to generate square caps. The current work also adds the option to provide a surface scaling parameter, similar to that available in CAD-based meshing, to support grid refinement studies.

IV. Volume Mesh Generation

This section provides a brief description of strand and Cartesian mesh generation strategies.

IV.A. Strand Mesh Generation

The strand mesh generation method consists of building several strand mesh layers, starting from the initial surface tessellation, and using the previous envelope surface as initial or base surface for each layer. Note that the number of strand layers is different from the number of points along each strand in the final mesh. The former is typically a small number below 5, and the latter is typically 50-100 depending on the resolution required. Two primary algorithms are used to generate each layer of the mesh– 1) `CLOVIS`, which stands for closest vertex in the isosurface of distance field, and 2) constrained elastic method for smoothing. Below is a brief summary of these methods. More details of these algorithms can be found in Refs. 15, 17.

1. Generate a thin initial strand layer with multiple strands emanating from vertices on very convex edges to improve mesh quality and coverage. Multiple strands are also generated at corners with visibility constraints.

2. Generate single-strand mesh layer, with thickness L, each strand joining a surface vertex to the closest point on the isosurface of distance field at L, I_L, using the `CLOVIS` method.

3. Smooth the top tessellation by solving a constrained elasticity problem. Each edge is assumed to be a zero-rest-length spring with stiffness inversely proportional to corresponding edge length on the lower tessellation, and each vertex is constrained to remain on or above the isosurface I_L.

4. Repeat steps 2 and 3 using the envelope surface as base for the next layer. The thickness for each layers is computed based on the desired number of layers and extent of the final strand mesh.

Note that most parts of the above algorithm are performed in a fully parallel manner except for the determination of the multiple strands at the necessary nodes. The `CLOVIS` method is done on a per-strand basis and is embarrassingly parallel. The constrained elastic smoothing method uses the nearest neighbor information and is also very amenable to parallelization.

Evaluation of realistic geometries using the available meshing techniques indicated that it is nearly impossible to generate meshes without self-intersections for realistic geometries, even simple ones such as a wing-body test case. Both local and non-local self-intersections can exist in realistic strand meshes. Local intersections occur between immediately neighboring cells and are caused by surface curvature, i.e. when the generating wall surface has highly concave areas. Non-local intersections are caused by topological constraints and geometric features that extend away from a main body.

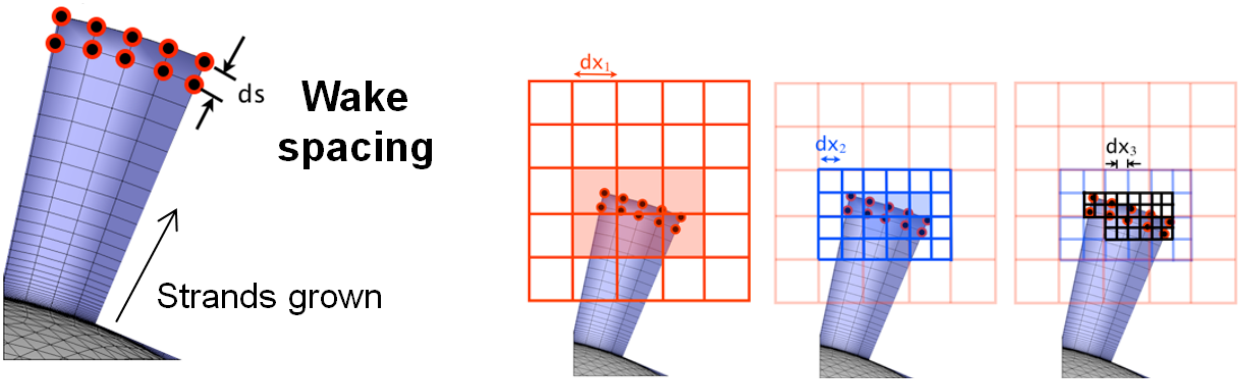
Both the inter-mesh and the intra-mesh connectivity in the strand grid framework are performed by a package called OSCAR,[12,15] which is developed by one of the co-authors. OSCAR utilizes the compact nature of strand grid to speed-up the overall domain connectivity process. Details of the algorithm can be found in Ref. 15.

IV.B. Cartesian Mesh

The Cartesian grid is generated following a Berger and Colella-style[36] multi-level block-structured AMR (SAMR) grid hierarchy. The coarsest level defines the physical extent of the computational domain and new levels are constructed from coarsest to finest. Each finer level is formed by selecting cells on the coarser level and then clustering the marked cells together to form block regions that will constitute the new finer level. The result is a hierarchy composed of nested refinement levels, with each level formed as a union of logically-rectangular grid regions.

The Cartesian grid is refined to match specified mesh spacing ratio, α, at the extents of the strand mesh, as demonstrated in Fig. 2. The location of the clip index elements (x, y, z) and the wake spacing Δs are provided to the Cartesian grid generator (a), cells on the Cartesian grid system that contain strand outer boundary elements are checked (b) to see whether the Cartesian grid spacing Δx is greater than the specified factor of the wake spacing $\alpha\Delta s$. If they are, $\Delta x > \alpha\Delta s$, the Cartesian cell is marked for refinement. All marked cells are clustered to construct a new finer level in the hierarchy. The process is repeated until no Cartesian cells are marked, fully satisfying overset donor-receiver requirements and ensuring good mesh overlap at the strand/Cartesian overset boundary.

After the initial generation of adaptive Cartesian grids to satisfy geometric requirements, grids are subsequently adapted throughout the simulation. Mesh cells that contain regions of swirling flow are identified using the scaled q-criteria scheme proposed by Kamkar et al.[37] Alternative solution quantities – e.g., vorticity



(a) Wake spacing at strand extents

(b) Cartesian refinement to match spacing

Figure 2. Off-body adaptive Cartesian mesh generation to resolve overset interface between strand and Cartesian grids.

or Q-criteria magnitude, density variation, entropy, etc. – could readily be substituted to drive refinement in the adaptive solution procedure. However, the scaled q-criterion automates the wake refinement procedure, which is an advantage over other quantities. If a defined quantity such as vorticity or Q criteria is used, predicting beforehand the appropriate threshold is often difficult. Also, the strength of the vortex wake may be variable throughout the simulation. Multiple runs are therefore required to determine a suitable threshold quantity. The scaled q quantity requires no such tuning and automatically adjusts to the changing scales of rotational flow in the wake.

IV.C. Automation of Volume Mesh Generation

Full automation of mesh generation is an important desired feature. The strand volume mesh generation requires the following inputs: 1) a surface tessellation, which can be composed of either triangles or quadrilaterals or a mix of both, 2) a strand length, which determines the outer extent of the strand grid, 3) a wall spacing to achieve a desired $y+$, and 4) a desired wake spacing, which is the normal spacing at the strand outer boundary. and 5) a ratio of the near-body to the off-body mesh resolution at the overset interface.

In an earlier work, current authors[38] analyzed the sensitivity of rotorcraft simulation to these input parameters and provided a best practice guideline to set these values. Based on the previous study, the default values of various input parameters are set as: 1) strand length $\rightarrow$ 0.4 $\times$ reference length used to compute the Reynolds number, 2) wall spacing $\rightarrow$ computed based on $y+ = 1$ at the flow Reynolds number, 3) number of normal strand points $\rightarrow$ 51, and 4) wake spacing $\rightarrow$ 0.08 $\times$ strand length. and 5) ratio of the near-body to the off-body mesh resolution $\rightarrow$ 1. The user can choose to either manually set the above listed meshing input parameters or they will be set to the default values. Other intrinsic parameters required for generating the strand mesh include the number of strand layers, the number of elastic smoothing iterations etc. These parameters are currently set based on heuristics to give a good quality volume mesh, but they will be improved further in the future. As mentioned before, the auto-generated strand mesh can self-intersect, and these intersections are automatically cleared during the simulation. The off-body Cartesian mesh is generated automatically and adapted after the strand mesh is generated. The entire volume mesh generation is performed in parallel at run time and takes anywhere from few seconds to minutes depending on the problem size and the number of processors used for the simulation.

V. Adaptive Surface Meshing

Adaptive mesh refinement (AMR) is an powerful tool to make better use of the available degrees of freedom to improve the solution accuracy. AMR also provides a way to capture more of the essential flow physics on a given computational resource. In general, AMR involves an increase in the number of grid cells in regions with significant flow or geometric features and a decrease in number of grid cells at uninteresting regions. Unlike a fixed mesh that is generated fully or partially agnostic of the flow solution, a solution-based adaptive mesh is intimately linked to the flow-field solution and alters as the flow field develops. Automatic adaptation of meshes based on flow solution also significantly reduces the need for human intervention.

The use of AMR is becoming more widespread in both the structured and unstructured grids. For

structured grids, hierarchical-type AMR techniques[31,36] have been well established. Note that, the off-body Cartesian mesh in the strand grid framework is already equipped with multi-level block-structured AMR capability for several years now. For the unstructured grids, metric-based mesh adaptation has gained broader application. Loseille et al.[39] demonstrated the potential of metric-based anisotropic mesh adaptation using several three-dimensional test cases. A review of the progress made in anisotropic mesh adaptation for CFD during the last decade is given by Alauzet and Loseille.[40]

The objective of this work is to develop an infrastructure that can support AMR in the near-body strand mesh. The strand mesh has a unique data structure that is unstructured on the surface and structured in the normal direction. This will allow the use of various unstructured grid adaptation techniques for surface mesh adaptation and the structured grid adaptation methodologies for the strand volume mesh adaptation. The initial focus is to achieve surface mesh adaptation within the strand grid framework. Future work will pursue volume mesh adaptation.

The mesh adaptation process can be decomposed into individual steps consisting of the flow solution, error estimation or flow feature detection, mesh size estimation, mesh generation and solution interpolation (see Fig. 3). An infrastructure is being developed under strand grid framework to support all these individual steps. Currently, all the steps except the error estimation and mesh size estimation step is incorporated.

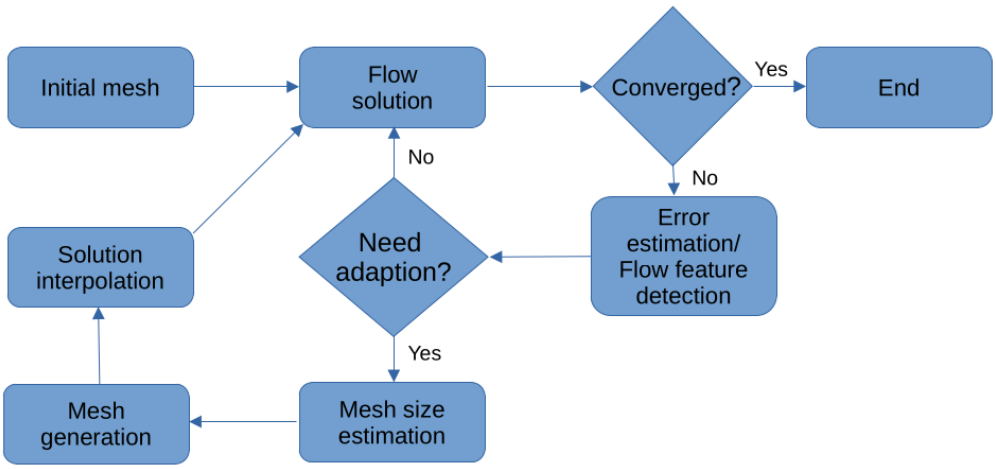


Figure 3. Solution-based mesh adaptation process.

The main strength of the strand grid framework is the fully automated parallel mesh generation process. In addition, the overset domain connectivity module (PUNDIT)[41] within Helios is extremely fast, fully parallel and has been very well tested. Therefore, the infrastructure for adaptive meshing is developed such that a new mesh is created every time the mesh needs to be adapted. PUNDIT is equipped to handle the interpolation between the old and the new meshes. Regenerating the mesh eliminates the possibility of ending up with poor quality mesh elements during adaptation process.

Prior to this work, the strand solver `mStrand` already had the ability handle multiple meshes. As a part of this work, the mesh dataset in `mStrand` is converted to a linked list instead of a fixed size array, to allow for easy addition and deletion of meshes. During the adaptation process, the new mesh is appended to meshes list and the old mesh is deleted when the solution transfer is completed. The adaptation process within strand grid framework consists of following steps:

1. `mStrand` computes an estimate of the error or detects flow features. Currently, the solver is equipped with a rudimentary shock sensor, which will be discussed later in the results section. Adaptation to error or adjoint quantities will be explored in future work.

2. `mStrand` determines the sizing field. This step is also under development.

3. A new surface mesh is generated using the sizing field information. The surface mesh on a CAD-based geometry is generated using Capstone and `bladeGen` is used for aerodynamic surfaces.

4. Both near-body and off-body volume meshes are generated as discussed in section IV.

5. `mStrand` appends the new mesh to its meshes list. In `mStrand`, all the meshes are partitioned to run across all the available processors. As a result, the solver is fully load balanced at all times. The new mesh is partitioned in a similar manner to run on all the processors. Next, all the memory associated with the flow solution on this new mesh is initialized.

6. `mStrand` passes the new mesh information to MELODI (Mesh-motion, Loading and Deformation Interface),[42] a fluid-structure interface designed to facilitate aero/structure exchanges for rotors. MELODI applies all the necessary motions and deformations to the new mesh. The new mesh is now positioned to receive interpolated solution from the old mesh.

7. PUNDIT receives the new mesh information and the solution is interpolated from the old mesh to the new mesh.

8. `mStrand` updates the solution of the new mesh using PUNDIT provided data. The old mesh is deleted from the meshes list and solution proceeds.

Note that all the steps are performed at runtime without any user intervention. Excepting the surface mesh generation, all other processes are done in parallel. Further, most of information transfer between different modules in Helios is done by exchanging python pointers. File I/O is used minimally to transfer some relatively small data.

VI. Results

The infrastructure developed for handling surface adaptation is tested for drag calculations ROBIN fuselage[43] and the transonic flow past ONERA M6 wing.

VI.A. ROBIN Fuselage

To demonstrate the ability to handle adaptive surface meshes on a CAD-based geometry, calculations are performed on the ROBIN helicopter fuselage. The ROtor Body INteraction (ROBIN)[43] is a generic helicopter fuselage geometry used for combined computational and experimental efforts to assess helicopter fuselage drag. The analytically-defined geometry was tested in the 2ft $\times$ 3ft Boundary Layer Channel Wind Tunnel located at NASA Langley Research Center. Model dimensions and tunnel flow characteristics are documented in Table 1.

Model Length L	28.235 in
Freestream Mach, M_∞	0.1 (34 m/s)
Reynolds Number, Re_L	1.6×10^6
Angle of Attack	0^o

Table 1. Operating conditions for ROBIN fuselage calculations.

In order to verify the solution using adaptive surface mesh, initial simulations are done using six different fixed meshes with varying surface tessellations. Note that the entire mesh system - the surface mesh, the strand volume mesh and the off-body Cartesian mesh is generated at runtime. The surface mesh is generated using Capstone. Two types of meshes are used - 1) Type A, that has uniformly spaced surface tessellation 2) Type B, that has a min/max ratio of 0.5, which provides refinement based on the curvature. Three different surface resolutions from coarse to fine is studied for both the mesh types. A summary of the characteristics of all the meshes is given in Table 2. To highlight the main difference between the type A and B meshes, Fig. 4 shows the surface tessellation of type A coarse, type B medium and type A medium meshes. Type B medium mesh has surface resolution similar to Type A medium mesh in curved regions and a resolution comparable to Type A coarse mesh in flat regions. Therefore, a type B mesh has fewer degrees of freedom compared to a type A mesh with corresponding resolution.

The volume meshing parameters are set to their default values in all cases. Figure 5 shows the auto-generated volume mesh for the type A fine grid setup. Cartesian refinement targets the outer boundary of the strand mesh, refining until the Cartesian spacing matches the cell size of the outer extents of the strand mesh.

Mesh Type	Resolution	Surface spacing	Min/max ratio	Surface mesh (faces)	NB vol. mesh (nodes)	OB vol. mesh (nodes)
A	Coarse	0.25 in	1.0	12.9K	0.33M	3.0M
	Medium	0.125 in		53.7K	1.33M	8.4M
	Fine	0.0625 in		211.0K	5.39M	24.1M
B	Coarse	0.5 in	0.5	7.4K	0.19M	3.0M
	Medium	0.25 in		24.7K	2.13M	8.4M
	Fine	0.125 in		83.4K	5.39M	8.4M

Table 2. Mesh statistics for ROBIN fuselage calculations; meshes generated automatically from CAD using prescribed inputs.

(a) Type A coarse mesh - 12.9K faces

(b) Type B medium mesh - 24.7K faces

(c) Type A medium mesh - 53.7K faces

Figure 4. Auto-generated surfaced meshes for ROBIN fuselage calculations.

In addition to the fixed grid simulations, a simulation with adaptive surface grid is performed. This simulation is started with the type B coarse mesh and adapted twice during the simulation to type B medium mesh and type B fine mesh, subsequently. The entire process including surface and volume mesh generation and solution interpolation from old to the new grid is done at runtime without any user intervention.

Calculations are performed in time-accurate mode using a time-step size of $\Delta t = 2.0 \times 10^{-5}$s. All the simulations are done for 15000 time-steps. When the simulation is performed with adaptive surface mesh, the adaptation is performed at time-steps 5000 and 10000. Within each time-step the mStrand cases converged the full l2-norm residual (including turbulence quantities) by one order-of-magnitude, which usually correlated to 4-6 dual-time subiterations per time-step. The flow is assumed to be fully turbulent with the Spalart-Allmaras turbulence model used by all solvers.

The computed drag coefficient (C_D) for all the cases are given in Table 3. Also, reported are the free-air calculated C_D using OVERFLOW and elsA flow solvers, both structured grid codes. As the grid is refined,

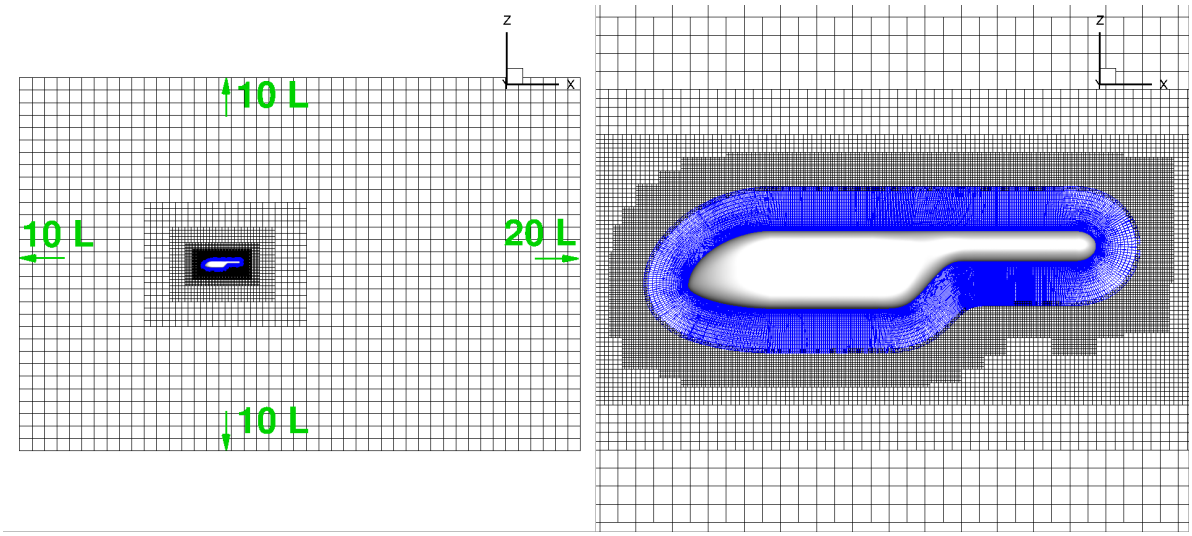


Figure 5. Dual-mesh used for ROBIN fuselage calculations.

Solver	Mesh Type	Resolution	C_D
mStrand	A	Coarse	0.154
		Medium	0.113
		Fine	0.108
	B	Coarse	0.185
		Medium	0.115
		Fine	0.107
	Adaptive	Varies	0.107
OVERFLOW free-air[43]	-	Fine	0.114
elsA free-air[43]	-	Fine	0.109

Table 3. Computed drag for ROBIN fuselage calculations.

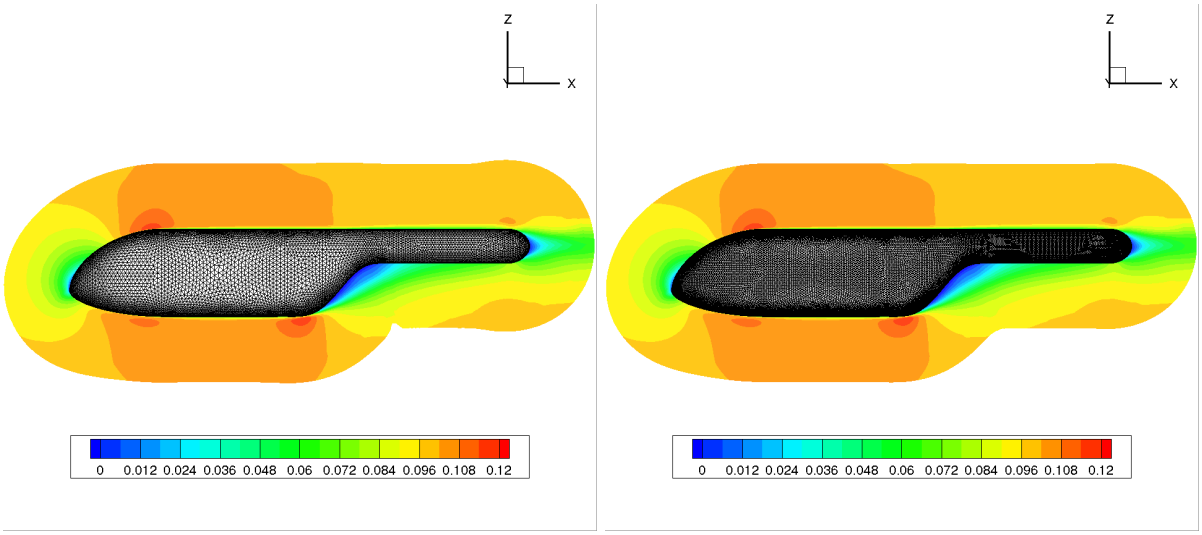
the drag value approach the OVERFLOW/elsA result, thus validating the overall solution procedure. Inspite of having lesser degrees of freedom, the type B fine mesh is able to predict a drag value comparable to the type A fine mesh. Further, the drag value predicted from the mesh with adaptive surface tessellation also converges to the expected value.

The solution interpolation from old grid to the new grid during adaptation is verified by plotting the u-velocity contours on mid-plane slice in Fig. 6. The plot shows the solution transfer from type B medium mesh to type B fine mesh at time-step 10000. The solution transfer appears to be visually clean without any obvious differences.

Figure 7 shows the convergence history of drag coefficient for all the cases using type B mesh as well as for the setup with adaptive mesh. Figure 7(a) gives a zoomed-out view and Fig. 7(b) gives a closer look for a better view of convergence of the adaptive mesh case. The fixed grid cases are converged well by the end of the simulation.The predicted drag for the adaptive mesh case is the same as the result using coarse mesh, as it should be, till the first adaptation. Following the adaptation, there is a jump in the drag value. Such a jump is expected because of drastic change to the mesh after adaptation. The drag values post-adaptation quickly gets close to the values obtained using the corresponding fixed grid simulation. Overall, the results show that the framework for handling runtime surface mesh adaptation is behaving as expected.

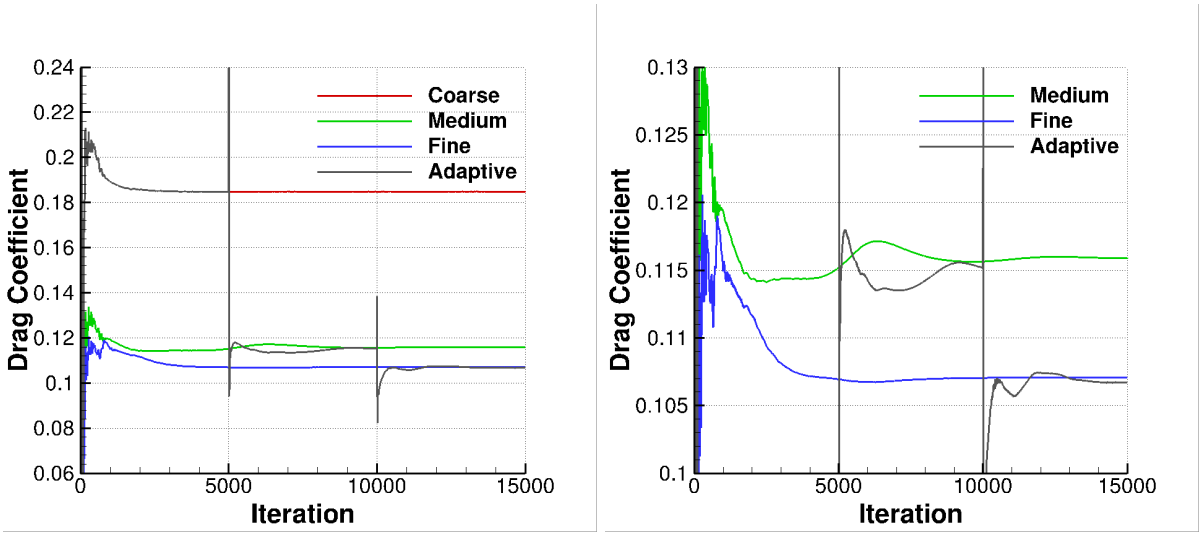
VI.A.1. Timing

Table 4 shows the timing for various adaptation stages. The simulations were done on DoD HPCMP "Onyx" HPC system, a Cray XC40/50 located at the Army Engineer Research and Development Center (ERDC) DSRC. Each node of Onyx contains Intel Xeon E5-2699v4 Broadwell processors with 44 cores per node, 128 GBytes of memory/node, and a clock rate of 2.8 GHz. The data presented in Table 4 is for the adaptation



(a) Original solution (b) Interpolated solution

Figure 6. Sectional velocity contours for ROBIN fuselage calculations



(a) Zoomed out (b) Zoomed in

Figure 7. Convergence history of drag coefficient for ROBIN fuselage calculations

No. of CPU cores	55	110	220	440	880
Surface mesh generation	15.04	15.12	15.22	15.14	15.10
Volume mesh generation	60.34	42.53	26.14	16.46	11.41
Data interpolation	1.30	0.76	0.52	0.54	0.64
Initialization	2.98	1.95	1.29	0.98	0.52
Total	79.66	60.04	43.17	33.12	27.67

Table 4. Timing (in sec) for various adaptation stages in ROBIN fuselage calculation.

from type B medium mesh to the type B fine mesh at time-step 10000. The timing results are shown on different number of CPU cores to understand the parallel scaling of the adaptation process. Note that the timing presented under initialization includes initialization of new mesh, deletion of old mesh and other small overhead costs. The surface mesh generation using Capstone is a serial process and therefore it's timing is independent of the number of CPU cores used. Both the volume mesh generation as well as the initialization stages show speed up with the core count, though their parallel efficiency is seen to reduce with the increasing

core count. The data interpolation process takes only a very small fraction of the total time taken in all core counts. This process shows improved timing with core count till 220 cores, after which it does not show any speed up. Clearly, the major portion of the total time is taken by the surface and volume mesh generation. At this point, the surface mesh generation from Capstone is the main bottleneck in getting faster adaptation process.

Comparing the above presented timing data with that for the flow solution will help understand the feasibility of the adaptation process. The near-body solver `mStrand` takes 0.32 sec per iteration to solve the type B medium mesh and 0.72 sec per iteration for the type B fine mesh, both run on the 220 CPU cores. Since the number of off-body grid points are very similar for both the medium and fine type B mesh setups, the off-body solution takes about 1.05 sec per iteration in both the cases, again on 220 cores. The time taken by the adaptation process can be estimated to be 20 to 50 flow solver time-steps. For test cases that do not require very frequent mesh adaptation, the cost associated with the adaptation process is quite reasonable, considering the potential savings using the methodology.

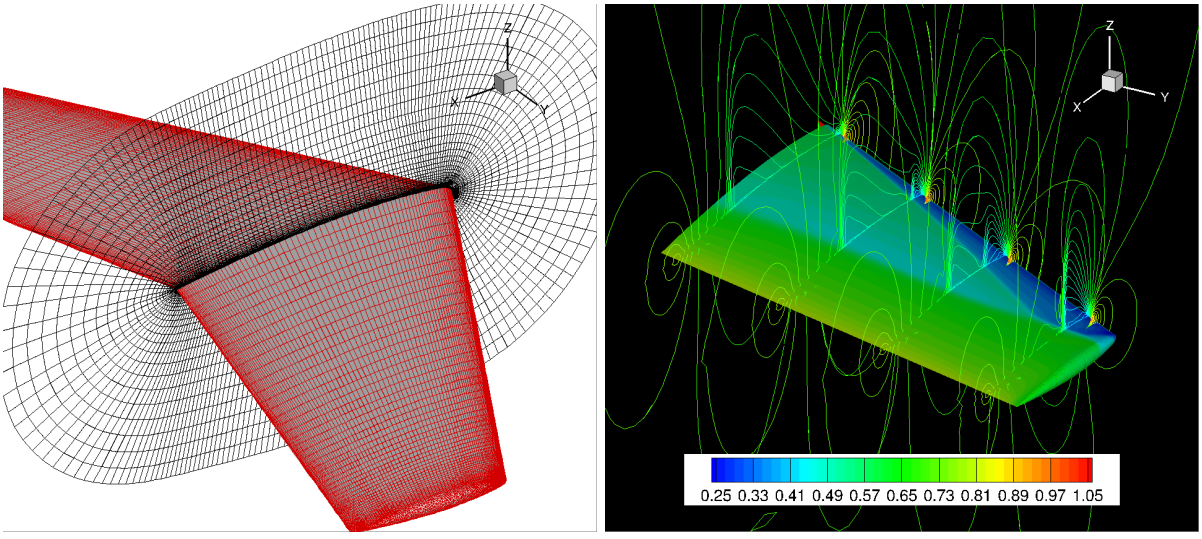
VI.B. Transonic Flow Past ONERA M6 Wing

An initial demonstration of a solution based surface adaptation using strand grid framework is provided using the transonic flow past ONERA M6 wing. The ONERA M6 wing experiment was originally described in an AGARD report by Schmitt and Charpin.[44] The experimental dataset has been widely used for CFD validation studies due to the simple geometry of the M6 wing with complex flow features characterized by the presence of lambda-shock on the top surface, which can be better resolved by clustering nodes around those locations. Details about the wing and the flow characteristics is provided in Table 5.

Root chord, c	0.806 m
Aspect ratio, AR	3.8
Taper ratio, λ	0.56
Freestream Mach, M_∞	0.84
Reynolds number, Re_c	14.6×10^6
Angle of attack	3.06^o

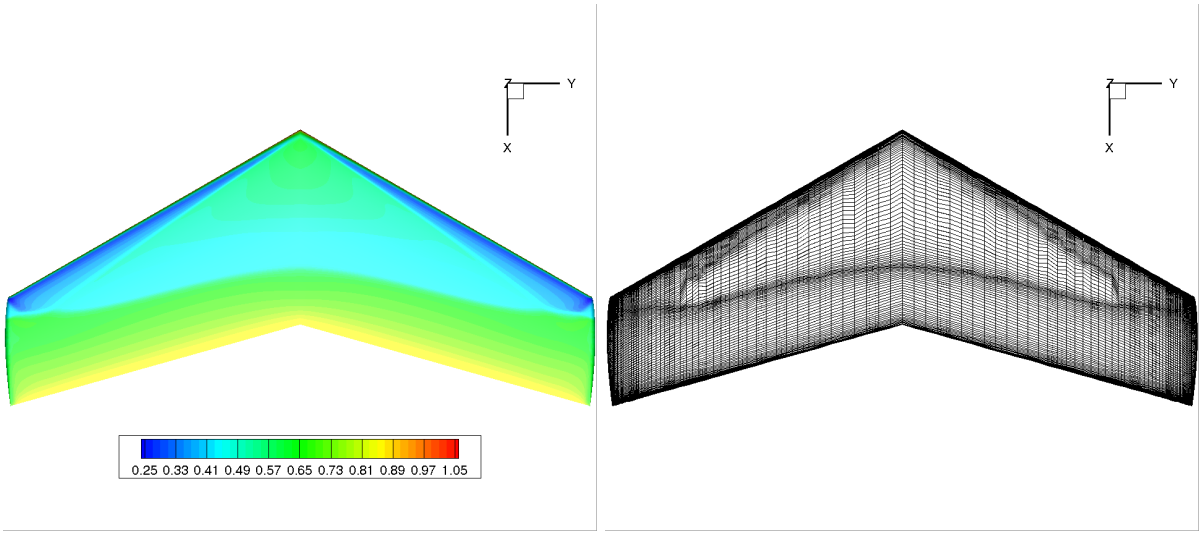
Table 5. Wing profile and operating conditions for ONERA M6 wing calculation.

The surface mesh is generated using the `bladeGen` utility at runtime. The baseline blade mesh has a total of $\sim$ 22.9K surface nodes with $\sim$ 20.2K quadrilateral elements and $\sim$ 5.5K triangular elements. The strand volume mesh goes out to $0.625c$ and has 61 nodes in the strand direction. The total number of strand nodes used is $\sim$ 1.4 million. Figure 8(a) shows the surface grid and a sectional view of the strand volume



(a) Grid (b) Sectional pressure contours

Figure 8. Mesh and sectional pressure for ONERA M6 wing simulation.



(a) Surface pressure (b) Adapted surface grid

Figure 9. Surface pressure and adapted surface grid for ONERA M6 wing simulation.

grid. A rounded tip cap is generated using the new feature added to `bladeGen`. The generated Cartesian off-body grid has 1.6 million nodes. The simulation is run as a steady calculation for 10000 time-steps with SA-RANS turbulence model in both the near and off-body solvers. Figure 8(b) shows the sectional pressure contours obtained using the fixed grid setup at different spanwise locations. The contours clearly show that the expected lambda-shock is captured.

In order to obtain a solution-based adapted surface mesh to capture the lambda-shock, the shock location is identified using pressure Hessian on the surface within the `mStrand` solver and this information is provided to the `bladeGen` utility. The `bladeGen` tool then redistributes various two-dimensional sectional grid to create a finer surface spacing around the shock locations. The total number of surface nodes remain exactly identical to the baseline grid along the span of the wing. However, the number of nodes in the root and tip cap region can be different. But, the total number of nodes will be roughly constant after adaptation.

The simulation with adaptive surface mesh is run for 10000 steps with the adaptation starting at time-step 5000 and thereafter the mesh is adapted every 1000 steps. Figure 9 shows the surface pressure contours predicted on the fixed grid as well as the final adapted surface grid. The surface mesh is seen to have adapted reasonably well to the shock location. However, one can notice some oscillations in the adapted grid shock front. Future improvements to the shock sensor would fix this issue.

Figure 10 shows the convergence history of the lift coefficient obtained from both the fixed and adaptive grid simulations. The fixed grid result converge to a steady value. The lift coefficient for the adaptive case also settle down fairly well to within 0.2% of the fixed grid value, but a small jump in the value is observed at every adaptation step. Note that the jump observed at the adaptation step is much smaller for this problem compared to the ROBIN fuselage case discussed in the previous section. The amount of jump is also seen to diminish as the solution progresses, but it does not fully go away. The shock location and the adapted grid were observed to change minimally through the end of the simulation. The issue could be again related to deficiencies in the shock sensor and would be rectified in the future. Nevertheless, the simulation demonstrates that the whole surface adaptation framework works as expected.

The improved shock capturing ability of the adapted grid is shown by plotting the sectional pressure coefficient from both the fixed and the adaptive surface mesh simulations in Fig. 11. Also, plotted is the experimental data provided in Schmitt and Charpin.[44] The result from the fixed grid simulation look quite satisfactory when compared to the experimental data. The result from the adaptive grid simulation looks almost identical to the fixed grid simulation. In order to highlight the differences, Fig. 12 provides a close-up view of the sectional pressure coefficient around the shock at two spanwise stations. The adapted grid can be seen to capture the gradients across the shock much better.

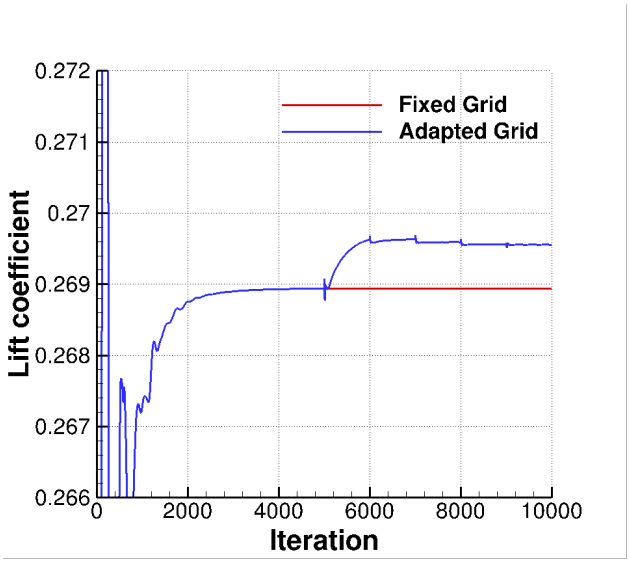


Figure 10. Convergence history of lift coefficient of for ONERA M6 wing simulation.

VI.B.1. Timing

Table 6 shows the average timing for various adaptation stages in the ONERA M6 wing simulation computed on 220 cores of DoD HPCMP "Onyx" HPC system. The timing per iteration for the near and off-body solution is also provided. The surface mesh generation using `bladeGen` is a serial process, similar to the methodology using Capstone. But, the cost associated with the surface meshing using `bladeGen` is much smaller. Therefore, the surface mesh generation process does not become a bottleneck to the parallel scalability of the adaptation procedure until the core count becomes really large. The time taken for the data interpolation as well as other overheads is relatively small as well. Comparing the time taken for the adaptation process with the that of the flow solution, the adaptation process completes in a wall-clock time equivalent to ~ 30 iterations, which is again very reasonable.

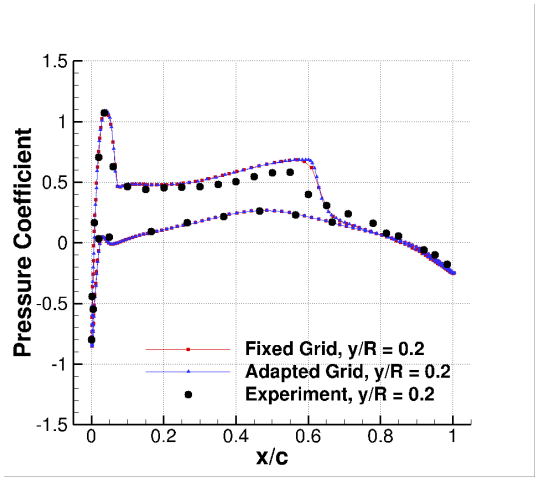
Process	Timing (in sec)
Surface mesh generation	1.52
Volume mesh generation	10.19
Data interpolation	1.34
Initialization	1.31
Total	14.36
Near-body solution (per step)	0.39
Off-body solution (per step)	0.10

Table 6. Timing for various adaptation stages and flow solution for ONERA M6 wing simulation on 220 cores.

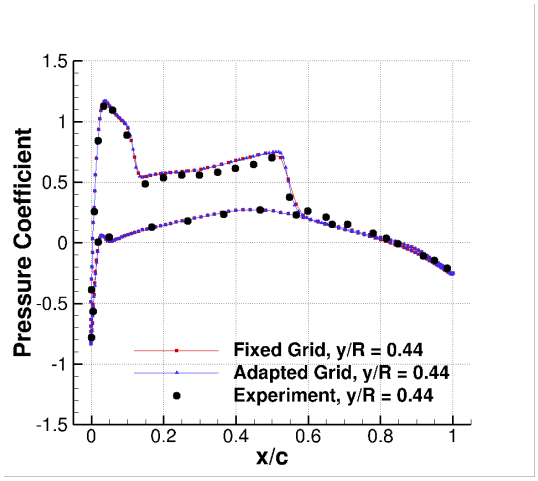
VII. Concluding Remarks

The ultimate goal of the HPCMP CREATE^{TM}-AV Helios framework is to achieve full automation of high-fidelity analysis capability and provide support during early stages of DoD's new rotary-wing aircraft acquisition programs. Helios employs a dual-mesh approach where a near-body grid resolves the boundary layer flow and the rest of the domain is covered using an adaptive Cartesian grid. The strand grid framework is being pursued as a viable approach to achieve automatic body conforming viscous mesh generation. Past work by current authors have made advances in strand volume meshing and domain connectivity to handle self-intersecting meshes that enable the strand technology to be used for very complex geometries. In addition, a production-oriented, fully parallel, highly efficient, and robust multi-strand solver called `mStrand` has been developed. More recently, an initial attempt was made towards automating the surface mesh generation for simple fuselages and rotor blades.

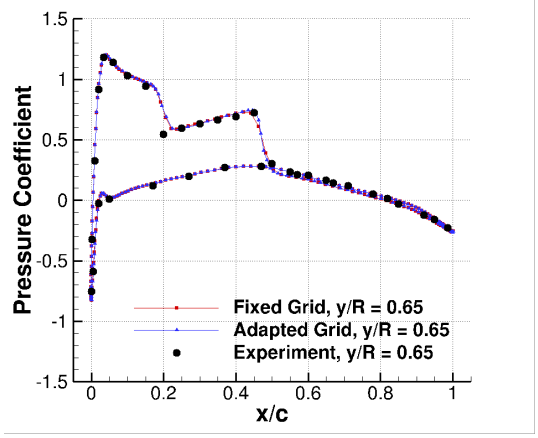
This work develops an infrastructure in the strand grid framework that can support fully automated surface mesh adaptation. Adaptive meshes allow better use of the available degrees of freedom, either to im-



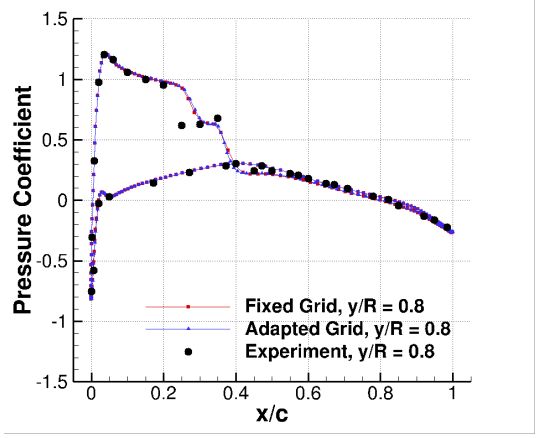
(a) $y/R = 0.2$



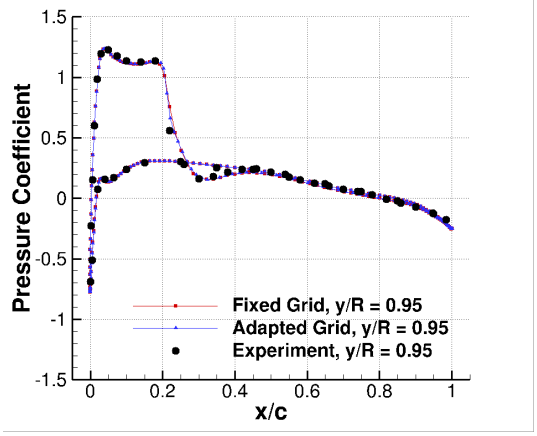
(b) $y/R = 0.44$



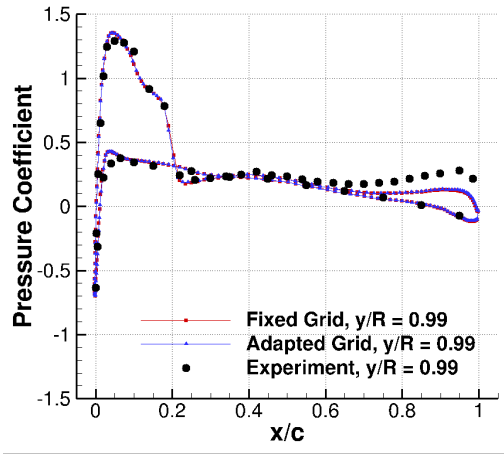
(c) $y/R = 0.65$



(d) $y/R = 0.8$

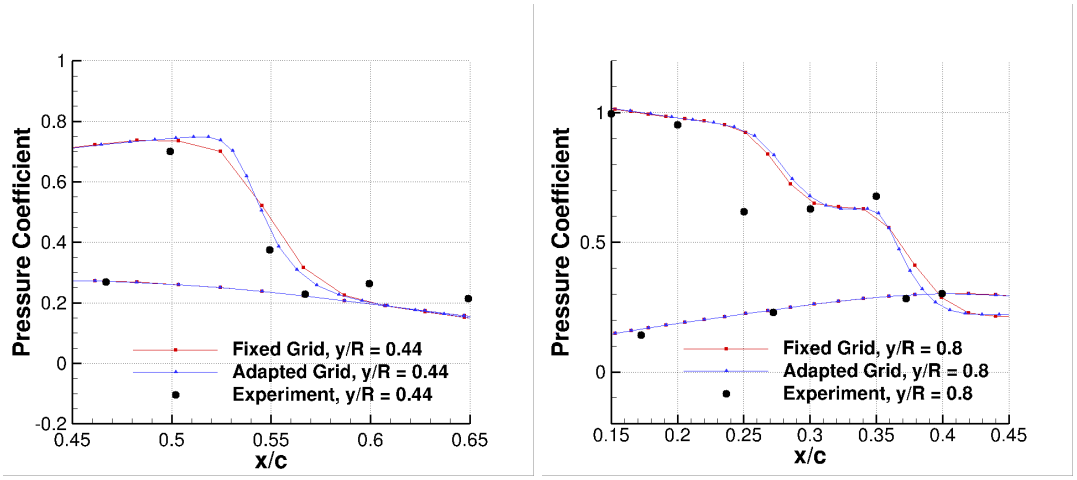


(e) $y/R = 0.95$



(f) $y/R = 0.99$

Figure 11. Comparison of sectional pressure coefficient at different spanwise location for ONERA M6 wing with the results from Schmitt and Charpin.[44]



(a) $y/R = 0.44$ (b) $y/R = 0.8$

Figure 12. Close-up view of sectional pressure coefficient around the shock at two spanwise location for ONERA M6 wing.

prove the solution accuracy provided a given amount of computational resource or reduce the computational cost required to attain a desired accuracy. The use of solution-based mesh adaptation in the near-body grid also improves the overall automation process by removing the need for any human knowledge about the flow solution. The developed infrastructure handles various stages of the surface adaptation process such as run-time generation of a new mesh, interpolation of solution from the old mesh to the new mesh and application of any underlying mesh motion/deformation. The framework ensures that the solution procedure continues seamlessly on the new mesh after adaptation without any human intervention. The framework does not yet support the step that determines the sizing field and our future work will focus towards that.

The developed framework was verified by performing uniform surface mesh refinement on the ROBIN fuselage. In addition, an initial demonstration of a solution-based surface mesh refinement was provided for the ONERA M6 wing configuration. Overall, the results show that the framework was able to handle runtime surface mesh adaptation as expected. The adapted grid demonstrated improved shock capturing ability for the ONERA M6 wing configuration. A parallel scaling study showed that all the processes, except the surface mesh generation, in the adaptation procedure show speed ups with increasing CPU core count. However, their parallel efficiency decrease as the core count increases. The data interpolation process was found to take a very small fraction of the total time. At this point, surface meshing, especially when starting from CAD, is the main bottleneck in getting faster adaptation process. Nevertheless, the adaptation process was found to take a wall-clock time of 20 to 50 flow solver time-steps in both the cases. For test cases that do not require very frequent mesh adaptation, the cost associated with the adaptation process is quite reasonable, considering the potential savings using the methodology.

As a closing remark, adaptive meshes clearly show promise towards complete automation of flow solution. In the future, the framework developed in this work will be equipped with the ability to determine surface mesh sizing either using a Hessian-based metric and/or an output-based error estimate and also with the capability to handle near-body volume mesh adaptation using a patch-based AMR technique.

VIII. Acknowledgments

Material presented in this paper is a product of the CREATE-AV Element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization Program Office (HPCMO). Some computer resources used for the validations were provided by the DoD HPCMO Frontier Program.

References

[1]Aftosmis, M.J., Berger, M.J., Alonso, J.J., "Applications of a Cartesian mesh boundary-layer approach for complex configurations," AIAA 2006-0652, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno NV, Jan 2006.

[2]Murman, S.M., Aftosmis, M.J., and Nemec, M., "Automated parameter studies using a Cartesian method," AIAA Paper 2004-5076, 22nd Applied Aerodynamics Conference, Providence RI, Aug 2004.

[3]Coirier, W.J., and K. G. Powell, "Solution-Adaptive Cartesian Cell Approach for Viscous and Inviscid Flows," *AIAA Journal*, Vol. 34, 1996, pp. 938-945.

[4]Nakahashi, K., "Building-Cube Method; A CFD Approach for Near-Future PetaFlops Computers," 5th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2008) Venice, Italy, June 2008.

[5]Peskin, C.S., "The Immersed Boundary Method," *Acta Numerica*, Cambridge University Press, Vol. 11, pp. 479–517, 2002.

[6]Tseng, Y.H., and J.H. Ferziger, "A Ghost-Cell Immersed Boundary Method for Flow in Complex Geometry," *Journal of Computational Physics*, Vol. 192, No. 2, pp. 593–623, 2003.

[7]Brehm, C., C. Hader, H.F. Fasel, "Novel Immersed Boundary/Interface Method for the Compressible Navier-Stokes Equations," AIAA-2012-1110, 50th AIAA Aerospace Sciences Meeting, Nashville, TN, Jan 2012.

[8]Berger, M, M. Aftosmis, S. Allmaras, "Progress Towards a Cartesian Cut-Cell Method for Viscous Compressible Flow," AIAA paper 2012-1301, 50th Aerospace Sciences Meeting, Nashville, TN, Jan 2012.

[9]Berger, M, M. Aftosmis, "An ODE-based Wall Model for Turbulent Flow Simulations," *AIAA Journal*, Vol .56, No. 2, pp, 700–714, 2017.

[10]Brehm, C., O. Browne, and N. Ashton, "Towards a Viscous Wall Model for Immersed Boundary Methods," AIAA 2018-1560, 2018 AIAA SciTech, Kissimmee FL, Jan 2018.

[11]Meakin, R., Wissink, A., Chan, W., Pandya, S., and Sitaraman, J., "On Strand Grids for Complex Flows", AIAA-2007-3834, 18th AIAA Computational Fluid Dynamics Conference, Miami, FL, June 2007.

[12]Sitaraman, J., Roget, B., and Wissink, A., "OSCAR - An Overset Grid Assembler for Overlapping Strand/Cartesian Mesh Systems", 11th Symposium on Overset Composite Grids and Solution Technology, Dayton, OH, October 2012.

[13]Lakshminarayan, V. K., Sitaraman, J., Roget, B. and Wissink, A. M., "Development and Validation of a Multi-Strand Solver for Complex Aerodynamic Flows," *Computers & Fluids,* Vol. 147, pp. 41-62, 2017.

[14]Lakshminarayan, V. K., Sitaraman, J., and Wissink, A. M., "Application of Strand Grid Framework to Complex Rotorcraft Simulations," *Journal of the American Helicopter Society,* Vol. 62, No. 1, pp. 1–16, 2017.

[15]Sitaraman, J., Lakshminarayan, V., Roget, B., and Wissink, A., "Progress in Strand Mesh Generation and Domain Connectivity for Dual-Mesh CFD simulations," 55th AIAA Aerospace Sciences Meeting, Grapevine, TX, January 2017.

[16]Lakshminarayan, V. K., Sitaraman, J., Roget, B., and Wissink, A. M., "Simulation of Complex Geometries Using Automatically Generated Strand Meshes," AIAA paper-2018-0028, 2018 AIAA Aerospace Sciences Meeting, Kissimmee, FL, January 2018.

[17]Roget, B., Sitaraman, J., Lakshminarayan, V., and Wissink, A., "Prismatic Mesh Generation Using Minimum Distance Fields," *Computers & Fluids*, Vol. 200, 104429, 2020.

[18]Lakshminarayan, V. K., Sitaraman, J., Jain, R. and Wissink, A. M., "Improvements to Automated Strand Meshing Capabilities for Rotary Wing Applications," Vertical Flight Society, Transformative Vertical Flight 2020, San Jose, CA, January 2020.

[19]Wissink, A., Potsdam, M., Sankaran, V., Sitaraman, J., and Mavriplis, D., "A Dual-Mesh Unstructured Adaptive Cartesian CFD Approach for Hover Prediction", *AHS Journal,* Vol. 61, No. 1, pp. 1–19, 2016.

[20]Roe, P.L., "Approximate Riemann Solvers, Parameter vectors, and Difference Schemes", *Journal of Computational Physics*, Vol. 43, pp. 357–372, 1981.

[21]Van Albada, G., Van Leer, B., and Roberts, W., "A comparative study of computational methods in cosmic gas dynamics", *Astronomy and Astrophysics* Vol. 108, pp. 76-–84, 1982.

[22]Spalart, P., and Allmaras, S., "A one-equation turbulence model for Aerodynamic flows", *La Recherche Aerospátiale*, Vol. 1, pp. 5–21, 1994.

[23]Menter, F. R., "Improved two-equation k-omega turbulence models for aerodynamic flows," NASA-TM-103975, Oct 1992.

[24]Menter, F. R, "Two-equation eddy-viscosity turbulence models for engineering applications," *AIAA Journal*, Vol .32, No. 8, pp, 1598–1605, 1994.

[25]Menter, F. R., Langtry, R. B., Likki, S. R., Suzen, Y. B., Huang, P. G., and Völker, S. "A correlation-based transition model using local variables Part I: model formulation," *Journal of Turbomachinery*, Vol. 128, No. 2, pp. 413–422, 2006.

[26]Medida, S. and Baeder, J. D. "Application of the correlation-based $\gamma - \overline{Re_{\theta t}}$ transition model to the Spalart-Allmaras turbulence model," AIAA-2011-3979, 20th AIAA Computational Fluid Dynamics Conference, Honolulu, HI, June 2011.

[27]Coder, J. G., "Development of a CFD-compatible transition model based on linear stability theory," Ph.D. Dissertation, 2014.

[28]Coder, J. G., Pulliam, T. H., and Jensen, J. C., "Contributions to HiLiftPW-3 using structured, overset grid methods," AIAA-2018-1039, 2018 AIAA Aerospace Sciences Meeting, Kissimmee, FL, Jan 2018.

[29]Saad, Y., and Schultz, M., "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, pp. 856–869, 1986.

[30]Karypis, G. and Kumar, V., "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, pp. 359-—392, 1999.

[31]Wissink, A., Hornung, R., Kohn, S., Smith, S., and Elliott, N., "Large-Scale Parallel Structured AMR Calculations using the SAMRAI Framework", Proceedings of Super-computing 2001 (SC01), Denver CO, Nov 2001.

[32]Wissink, A., Sitaraman, J., Jayaraman, B., Roget, B., Lakshminarayan, V., Potsdam, M., Jain, R., Leffell, J., Forsythe, J., and Bauer, A., "Recent Advancements in the Helios Rotorcraft Simulation Code", AIAA-2016-0563, 54th Aerospace Sciences Meeting, San Diego, CA, January 2016.

[33]Leffell, J., Sitaraman, J., Lakshminarayan, V., and Wissink, A., "Towards Efficient Parallel-in-Time Simulation for Periodic Flows", AIAA-2016-0066, 54th Aerospace Sciences Meeting, San Diego, CA, January 2016.

[34]Mestreau, E., R. Aubry, S. Dey, and M. Richarson, "HPCMP CREATE-AV Kestrel New Capabilities and Future Directions," AIAA 2019-1716, 57th AIAA Science and Technology Forum, San Diego, CA, January 2019.

[35]Subramanian, G and Raveendra, V., V., S., and Kamath, M. G., "Robust boundary triangulation and Delaunay triangulation of arbitrary planar domains," *International journal for numerical methods in engineering*, Vol. 37, No. 10, pp. 1779–1789,1994.

[36]Berger, M., and Colella, P., "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *Journal of Computational Physics*, Vol. 82, pp. 65–84, 1989.

[37]Kamkar, S., Wissink, A., Sankaran, V., and Jameson, A., "Feature-Driven Cartesian Adaptive Mesh Refinement for Vortex-Dominated Flows," *Journal of Computational Physics*, Vol. 230, No. 16, pp. 6271–6298, 2011.

[38]Lakshminarayan, V. K., Sitaraman, J., and Wissink, A. M., "Sensitivity of Rotorcraft Hover Predictions to Mesh Resolution in Strand Grid Framework," *AIAA Journal*, Vol. 57, No. 8, pp. 3173–3184, 2019.

[39]Loseille, A., Dervieux, A., and Alauzet, F., "Fully Anisotropic Goal-Oriented Mesh Adaptation for 3D Steady Euler Equations,", *Journal of Computational Physics,* Vol. 229, No. 8, pp. 2866–2897, 2010.

[40]Alauzet, F., and Loseille, A., "A Decade of Progress on Anisotropic Mesh Adaptation for Computational Fluid Dynamics," *Computer-Aided Design,* Vol. 72, 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation, pp. 13–39, 2016.

[41]Sitaraman, J., Floros, M., Wissink, A., and Potsdam, M., "Parallel domain connectivity algorithm for unsteady flow computations using overlapping and adaptive grids," *Journal of Computational Physics*, Vol. 229, No. 12, pp. 4703–4723, 2010.

[42]Roget, B., Sitaraman, J. and Wissink, A., "Maneuvering Rotorcraft Simulations Using CREATE^{TM}-AV Helios," AIAA-2016-1057, 54th AIAA Science and Technology Forum and Exposition, San Diego, CA, January 2016.

[43]Schaeffler, N. W., Allan, B. G., Lienard, C., and Le Pape, A., "Progress Towards Fuselage Drag Reduction via Active Flow Control: A Combined CFD and Experimental Effort," 36th European Rotorcraft Forum, Paper 064, Paris, France, September 2010.

[44]Schmitt, V., and Charpin, F., "Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers," Experimental Data Base for Computer Program Assessment, Report of the Fluid Dynamics Panel Working Group 04, AGARD AR 138, May 1979.