Dynamic Adaptive Mesh Refinement for WMLES on Complex Configurations

Dimitri J. Mavriplis,[*] Zhi Yang
Scientific Simulations LLC, Steamboat Springs, CO 80487, USA

Andrew C. Kirby
University of Wyoming, Laramie, WY 82071, USA

This paper presents a dynamic adaptive mesh refinement (AMR) capability for wall-modeled large-eddy simulations (WMLES) of complex configurations using body-fitted, unstructured grids with hybrid element types. The approach links legacy computational fluid dynamics (CFD) codes to a highly scalable octree-based mesh-subdivision library capable of handling tetrahedral, prismatic, pyramidal, and hexahedral elements. By defining and implementing a general application programming interface (API), the framework enables large-scale dynamic AMR on high-performance computing (HPC) systems without requiring internal modifications to existing CFD codes. The development of the AMR interface is described in detail and demonstrated by coupling it with two CFD solvers, NSU3D and FUN3D. Results are presented for canonical test cases as well as RANS and WMLES applications of industrial relevance, including efficiency and scaling performance. The paper concludes with a discussion of future extensions, including support for GPU-accelerated architectures.

I. Introduction

The objective of this work is the development and demonstration of a dynamic adaptive mesh refinement (AMR) capability for wall-modeled large-eddy simulations (WMLES) of complex configurations using arbitrary grid types, including unstructured grids with hybrid element types. Our approach consists of linking legacy computational fluid dynamics (CFD) codes to an octree subdivision library known as t8code,[1] which is highly scalable and can handle all element types, including tetrahedra, prisms, pyramids, and hexahedra. Through the definition and implementation of a suitable API, our approach enables dynamic AMR at scale on high-performance computing (HPC) systems with no internal changes to existing CFD codes. Although many WMLES applications can be considered as statistically steady or quasi-steady problems for which an optimized final static grid can be constructed either manually or through repeated application of AMR, the development of a fully dynamic AMR capability offers several advantages. Firstly, an efficient dynamic AMR capability enables more frequent mesh refinement passes during the simulation phase, opening up new possibilities in the use of refinement criteria targeting time-varying spatial error, while at the same time retaining all the capabilities of quasi-steady AMR approaches, including techniques for targeting numerical and modeling error separately.[2] At the same time, the tightly coupled nature of the dynamic AMR library provides a more streamlined and simpler-to-use workflow, which can be run seamlessly on large-scale HPC architectures, including heterogeneous GPU hardware. Finally, we anticipate that truly dynamic AMR capabilities will be required for grand challenge problems and certification by analysis, which involve multiple time scales, such as dynamic maneuvers and/or relative body motion, in addition to the unsteady nature of scale-resolved turbulence eddies. In previous work, we have developed experience with dynamic AMR for hybrid RANS-LES (HRLES) applications. Some of this initial work was performed over a decade ago on the HELIOS rotorcraft code,[3] which used an off-body Cartesian mesh with patch-based subdivision dynamic AMR through the SAMRAI library.[4] Subsequently, we developed an alternate approach based on octree subdivision using the p4est library,[5] initially developed at the University of Texas at Austin, which was shown to be significantly more efficient and scalable than the patch-based approach. Although our initial p4est implementation used the dynamic AMR capability on simple Cartesian meshes with high-order DG discretizations,[6] more recently we have extended this capability to include curved and mapped element body-fitted meshes for

[*]Corresponding author: mavripl@scientific-sims.com

HRLES and WMLES.[7] However, as with most subdivision AMR libraries, p4est is limited to the use of hexahedral element meshes and is thus not suitable for near-body meshes and complex configurations.

t8code is a new dynamic element-subdivision AMR library being developed as a successor to p4est, which provides support for arbitrary element types.[8] t8code is an open source library,[9] supported by DLR in Germany, which incorporates all of the p4est functionality for hexahedral elements, and adds support for tetrahedral, prismatic, and pyramidal elements. Both p4est and t8code use the forest of octrees paradigm. In this approach, an initial coarse unstructured mesh is read in, and each element of the mesh is used to define the root node of an octree, which enables recursive subdivision of that element and its refined children, as illustrated in Figure 1. In p4est, the initial forest of octrees was not distributed, whereas in t8code, the octree forest is fully distributed across all processors, enabling better scalability and the use of significantly finer initial meshes, as is required for complex configurations. As with p4est, t8code is designed to efficiently manage distributed adaptive meshes according to user-supplied refinement and coarsening criteria. The library performs mesh refinement, repartitioning, and load-balancing, while ensuring 2:1 refinement interfaces and managing ghost or halo inter-processor MPI communication schedules.[10] Other capabilities are available, such as fast search algorithms for point-in-cell inclusion tests. t8code has been demonstrated on up to one million MPI ranks and over 1 trillion mesh elements, targeting exascale applications on leading-edge hardware.[11]

Although p4est and t8code offer various functionalities, our approach consists of using the minimum information from these AMR libraries that is needed to reconstruct the specific data structures and information required for the target CFD codes. Generally, this consists of using only mesh topology and MPI communication patterns determined by the AMR library. For example, given the mesh topology, different data structures are required for cell-centered versus node-centered discretizations. Similarly, during a regridding and load re-balancing operation, the cell address correspondence between old and newly distributed grids is used to build the appropriate MPI communication buffers, which are then used for the solution-data transfer. Minimizing our API dependence on the AMR library in this way enables offloading these additional operations to GPU architectures in future work, where the paradigm consists of having the dynamic AMR library manage the mesh topology on the host CPU cores while all other operations, including flow solution, reside on GPUs, and direct GPU-to-GPU communication is used. Additionally, this strategy will facilitate the use of alternate AMR libraries within our framework in the future.

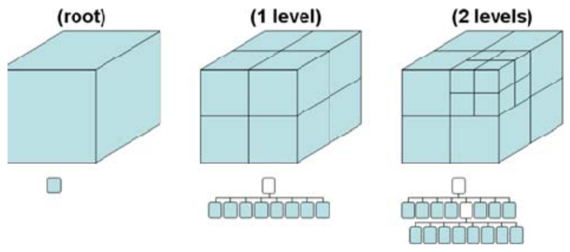


Figure 1. Illustration of octree refinement strategy for hexahedral element meshes.

The development of element-subdivision AMR methods[9,12] has progressed somewhat separately from anisotropic local[13] and global[14–17] remeshing work, which has been prevalent in the aerospace CFD community, particularly for RANS applications. On the one hand, dynamic AMR applications of the subdivision approach, until recently, have been mostly confined to Cartesian or hexahedral meshes, which have limited its applicability. At the same time, subdivision approaches, which mostly produce isotropic refinement patterns, have generally been concerned with the efficiency and scalability of the mesh refinement mechanics. For WMLES applications, the use of anisotropic refinement is arguably less important, with the preservation of near-isotropic cells seen as a desirable feature. Additionally, the efficiency and scalability of the method take on additional importance for dynamic applications. On the other hand, the anisotropic refinement community has made significant advances in mesh refinement criteria, error estimation, and the robust inclusion of geometry data for CFD applications. The objective in this work is to combine the best of both approaches into a framework that is suitable for WMLES applications for complex configurations using body-fitted unstructured meshes.

A. Adaptive Mesh Refinement Interface Development

1. Approach

Here, we focus on the implementation of dynamic AMR through the development of our interface for linking legacy CFD codes with the t8code AMR library. In general, our approach is to use the minimum information necessary from the AMR library and reconstruct all required information in the interface using the native flow solver routines based on the input from the AMR library. For example, the AMR library is capable of providing face normals, cell volumes, and other metrics, although we avoid using these facilities to ensure that the flow solver reproduces identical results on a static mesh, whether used directly or through the AMR library. The basic information required from the AMR library includes lists of cells on each partition, with inter-partition communication schedules (i.e., real to ghost cells), as well as vertex coordinate information. We consider AMR interface implementations for cell-centered discretizations as well as vertex-based discretizations. The implementation is more complicated for vertex-based discretizations due to the fact that the t8code AMR library does not include any concept of global vertex addresses. This is illustrated in Figure 2, with a simple example of quadrilateral cells in 2D that are split across two partitions. In this case, each cell appears once in a particular partition. However, at partition boundaries, vertices that belong to cells on both sides of the partition are replicated in each partition. t8code, as well as many other cell-based AMR techniques, do not include information on which vertices are replicated and do not include the notion of real (i.e., dominant) and ghost vertices. Therefore, additional implementation is required to establish these qualifiers, using cell-to-vertex information and inter-partition cell transfers (i.e., real-cell to ghost-cell MPI communication).

For cell-centered discretizations, non-contiguous refinement boundaries can be handled in a straightforward way using the native face-based fluxes inherent in these discretizations, as illustrated in Figure 3. However, for vertex-based discretizations, additional complications arise at non-conforming refinement interfaces. These can be treated in two ways. The first involves modifying the discretization in the CFD code to explicitly take into account these non-conforming topologies. The second approach involves the use of transition elements, where large cells at interface boundaries are refined anisotropically in a manner that ensures contiguous faces across all elements in these regions. This approach is adopted in this work, in the interest of providing a software interface that requires no changes to legacy CFD solvers.

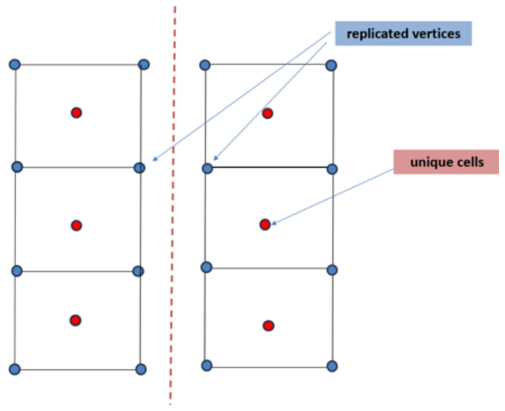


Figure 2. Illustration of cell-based partitioning used by t8code AMR library with replicated vertex values.

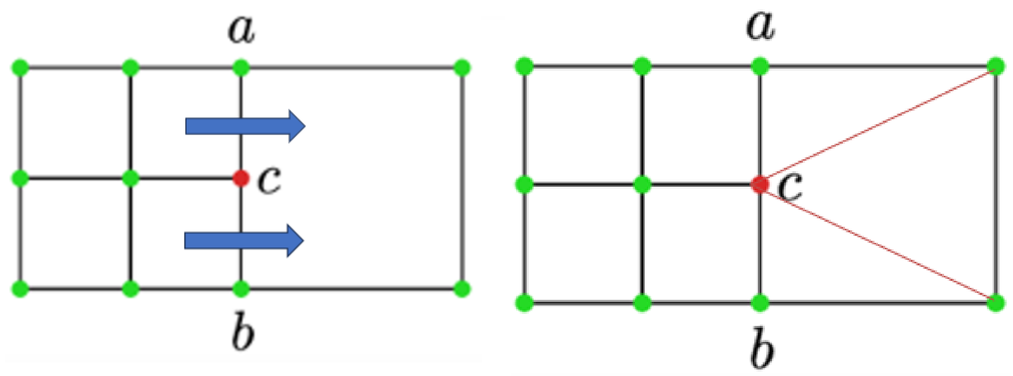


Figure 3. Illustration of non-conforming refinement boundary using (left) face-based fluxes at interface for cell-centered discretizations and (right) transition elements for vertex-based discretizations.

In using transition elements to merge together the refined and unrefined cells on either side of the refinement boundaries, we assume no greater than 2:1 refinement jumps at non-contiguous interfaces, which can be guaranteed by the t8code refinement library. The principal idea is to split unrefined elements at these boundaries into topologies that include only known element types (i.e., tetrahedra, prisms, pyramids, and hexahedra) and result in fully contiguous meshes with no overlap or non-matching faces. In this manner, the CFD code can proceed on the resulting mesh without the need for any internal modifications.

For the creation of transition elements, only a specific number of cell refinement patterns are permitted for each cell type. For tetrahedral elements (the simplest case), only three types of refinement are permitted, i.e., two transitional element refinement types, and a full 8:1 refinement type, as shown in Figure 4. Thus, if a tetrahedron has one face that lies on a non-contiguous refinement boundary, that element is subdivided following the 1:4 pattern in the figure, resulting in a conformal face topology at that boundary. However, elements may exist that are not refined but include an edge that is refined by a neighboring element subdivision. In this case, the tetrahedron can be subdivided into two smaller tetrahedra, as a 1:2 subdivision, as shown in the figure. On the other hand, if a tetrahedron is found that has two edges that do not share a face that are subdivided, then there is no compatible transition element subdivision possible, and the element is flagged for full 8:1 refinement. For other element types, there are more permitted subdivision types, although these are still restricted to those shown in Figures 5, 6 and 7 (for hexahedral, prismatic and pyramidal elements, respectively) which are determined by the cell edges that must be refined.

One of the aspects of this approach is that each time an element is flagged for refinement into transition elements (or full refinement if incompatible refinement is detected), new additional edges on these cells may be flagged for refinement. Since each edge is shared by a variable number of elements, these in turn affect the refinement pattern of neighboring elements. Thus, the process of determining a compatible refinement pattern is iterative. Multiple passes must be made through the mesh until no new edges are refined and a compatible refinement pattern is obtained. At this point, the transition elements can be inserted, and the resulting mesh is guaranteed to be fully contiguous.

It is important to note that the transition elements are invisible to the AMR library, which only accounts for fully refined elements and is agnostic about the creation of any transition elements. The transition elements are discarded prior to the next adaptive refinement operation, and are never further subdivided themselves in the AMR process, thus avoiding the generation of poorly shaped elements which could result from the subdivision of transition elements.

Note that his approach is based on edge refinement rather than cell refinement, whereas the AMR library does not maintain any edge data structures. The implementation of compatible refinement patterns and the creation of transition elements are performed in a two-level approach, where the t8code AMR library is responsible for all fully refined elements, and the interface routines are responsible for establishing compatible refinement patterns and inserting transition elements. In a first pass, the flow solver determines a set of cells to be refined or de-refined, and sends this cell-refinement map to the t8code AMR library, which returns the refined mesh, which also enforces 2:1 refinement boundaries. Next, the interface routines extract the edges of the mesh and identify edges that are split at refinement boundaries. Cells with incompatible edge refinement patterns are then flagged for full refinement. This is done by flagging all edges of the cell for refinement.

Once a compatible refinement pattern has been determined, the t8code library is called again to refine the additional cells marked for full refinement, and the transition elements are inserted into the resulting mesh. Due to the requirement of enforcing 2:1 refinement boundaries, it is possible that the mesh returned from the library contains additional refined cells, and a new compatible refinement pattern may need to be established. Thus, the procedure may be repeated until a compatible edge refinement pattern is established that requires no new fully refined elements.

On the first level of refinement, at most a single additional pass may be required, whereas with further refinement levels, more passes are possible. In practice, only 2 or 3 passes are usually required to obtain a compatible edge-refinement pattern with multiple refinement levels. This was the case in the original work on transition elements performed by the first author in reference,[18] and has been found to hold for most cases tried to date in this project. We note that our interface can be extended to determine a compatible refinement pattern that enforces 2:1 refinement boundaries on its own, resulting in no additional passes through the t8code library, although this has not been pursued as the additional complication was not deemed necessary.

2. *Incorporating CAD-consistent geometry*

A capability for positioning newly refined surface mesh points on the original CAD geometry definition during the AMR process has also been developed and is illustrated herein. For CFD meshes generated with the Heldenmesh software, a projection tool has been developed by Helden Aerospace Inc. to project each newly created surface mesh point onto the geometry definition used by the Heldenmesh software in the generation of the original coarse mesh. In the AMR process, initial refined surface mesh point coordinates are computed using linear interpolation from surrounding coarser-level mesh points. All new surface mesh points created on each MPI rank are then gathered

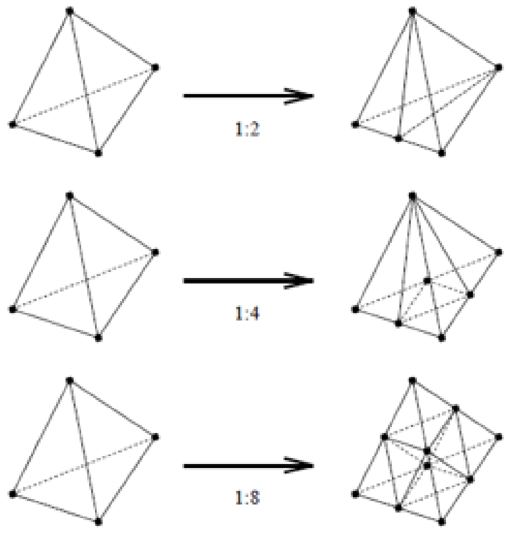


Figure 4. Permitted subdivision types for tetrahedral elements.

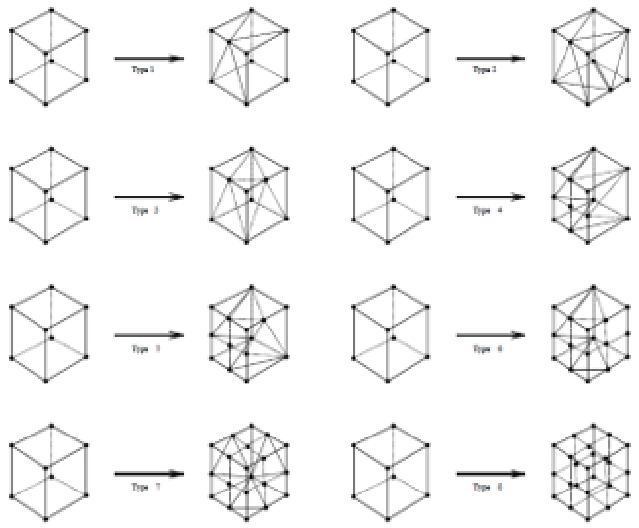


Figure 5. Permitted subdivision types for hexahedral elements.

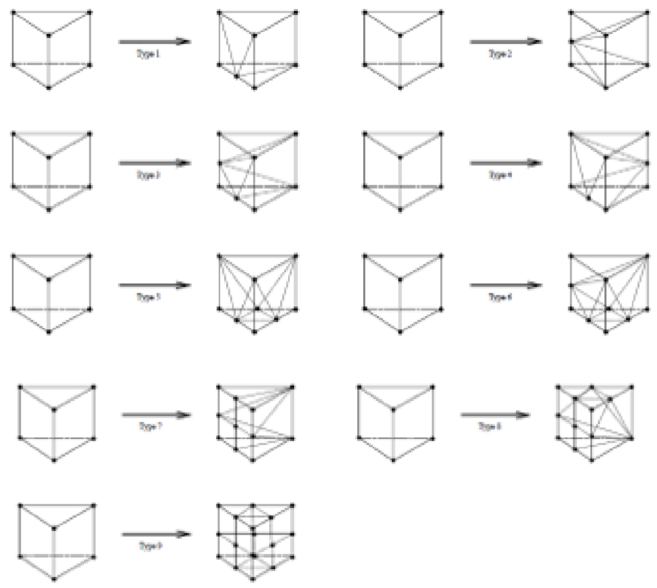


Figure 6. Permitted subdivision types for prismatic elements.

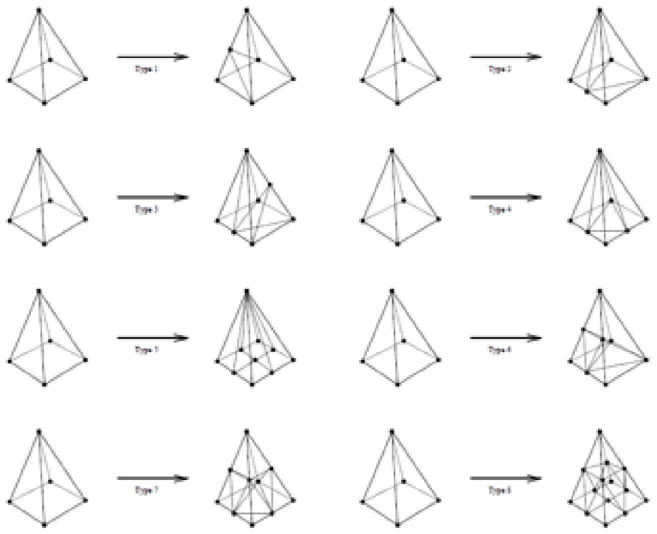


Figure 7. Permitted subdivision types for pyramidal elements.

and written to a file. Each MPI rank then executes an instance of the Helden projector code, which reads in the corresponding list of points to be projected and, in turn, writes out a new list of surface point coordinates. These new coordinates are then read back in by the corresponding MPI rank in the AMR interface. These new coordinates replace the original values obtained by linear interpolation in the newly refined AMR mesh.

In addition to the initial point coordinates, the mesh point files supplied to the Helden projector kernel contain one or two patch numbers for each point, corresponding to the geometry patch onto which the point should be projected, and a normal vector, which defines the direction of projection. The projection algorithm is illustrated in Figure 8, for a case with a coarse initial mesh in the presence of high curvature CAD geometry. An initial approach that locates the closest normal distance point on the CAD geometry was found to be inadequate for these types of configurations, as shown in the middle figure, whereas the use of the face normal projection direction was found to be more accurate and robust, as shown in the right figure. Inclusion of the CAD surface patch ID in the input file ensures that the mesh point is projected to the correct corresponding CAD surface. For cases where a new point resides on the boundary of two patches, both patch IDs are listed, ensuring that the new point lies exactly on the delimiting or trimming curve between these two patches as defined in the CAD geometry.

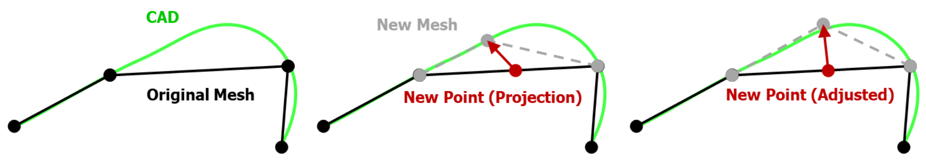


Figure 8. Surface point projection onto CAD consistent geometry

Once the surface mesh points have been projected to the CAD surface geometry, the positions of the interior CFD mesh points must be smoothed in order to guarantee a consistent mesh. This is achieved by invoking a previously developed mesh deformation solver.[19] This code has been added to the AMR interface as a callable component that takes as input a mesh with original and displaced coordinates, and returns a set of new smoothed mesh coordinates. The mesh deformation code is based on a linear elastic analogy, which treats the interior mesh as an elastic material that deforms in response to the applied surface displacements, using a variable modulus of elasticity that can be prescribed as inversely proportional to the local cell volume, thus avoiding the creation of small and/or negative volume cells in the final AMR mesh.

A demonstration of the surface point projection approach is provided using the HLPW5 Case 1 geometry.[20] The initial mesh consists of a WMLES mesh generated by Helden Aerospace Inc., containing approximately 5.6 million vertices. This mesh corresponds to a uniformly coarser version of the workshop mesh labeled *h5c1_xc_1.lb8.ugrid* available on the workshop website (mixed-element unstructured, 39.3 million points). Although the initial mesh contains a relatively fine surface discretization, small geometric features in the region of the wing-tip trailing edge are poorly resolved on this mesh, as shown in the coarse mesh detail in Figure 9. The figure illustrates the progressive refinement from the initial mesh through two full refinement passes, showing how the refined meshes gradually improve the approximation of the surface geometry in the wing tip region. In this example, for the finest mesh running on 400 CPU cores, 13 million surface points were projected in 10 seconds of wall clock time.

3. *AMR Parallel Performance*

In this section, we examine the parallel performance of the AMR process for full refinement of a complex geometry. The test case uses the HLPW5 Case 2.4 configuration,[20] a full detailed high-lift geometry shown in Figure 10. This configuration is substantially more complex than the Case 1 configuration, with 655 surface patches in the Heldenmesh surface definition. The initial mesh is a WMLES mesh generated by Helden Aerospace Inc., containing approximately 19.4 million vertices. It corresponds to a uniformly coarser version of the workshop mesh labeled $h5c21\_wmles\_xc07.b8.ugrid$ (mixed-element unstructured, 76 million points) available on the workshop website. Two full-refinement passes are applied to this mesh, producing meshes with approximately 153 million and 1.2 billion points, respectively. The wall-clock times for the total and individual refinement stages are summarized in Table 1, for runs executed on 400, 3200, and 6400 cores. For each configuration, the total refinement time is compared against the cost of a representative CFD solver cycle (a single point-implicit RK3 step and the solution of a BDF2 time-implicit step using 25 linear solver subiterations). These results show that the t8code refinement library is extremely fast, accounting for only a small fraction of the total refinement time. The subsequent data-structure construction phase scales

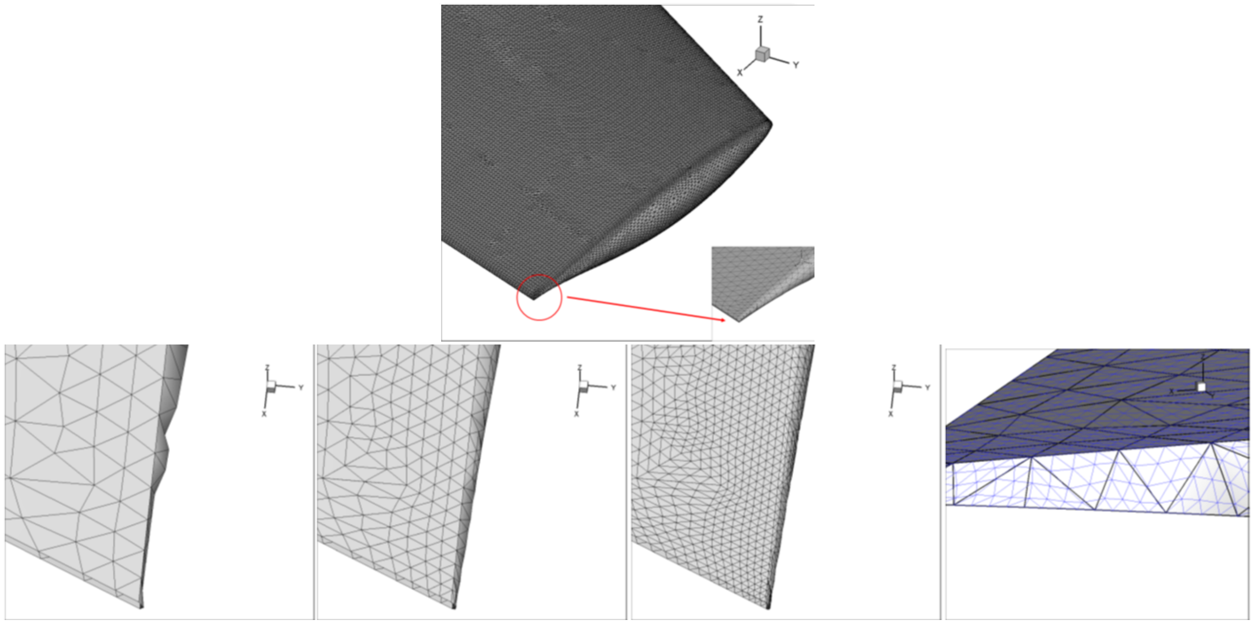


Figure 9. Illustration of CAD-consistent mesh refinement for HLPW5 Case 1 test case highlighting recovery of CAD geometry with increasing refinement levels in wing-tip trailing edge region and (far right) overlay of initial and final refined surface mesh.

approximately linearly with grid size and inversely with the number of processor cores. However, the absolute time required in this phase is not yet optimal, and work is ongoing to improve its overall efficiency. The CAD-projection phase exhibits significant processor-to-processor variability. In Table 1, the first number reports the time on processor 0, while the second number reports the global wall-clock time required for all processors to finish. The discrepancy reflects load imbalance arising from the highly uneven distribution of surface grid points among processors. The total refinement time also includes additional required operations—such as CFD-specific data-structure generation and mesh-consistency checks—but excludes the mesh-smoothing process. The latter contributes roughly 20–40% of the total refinement cost for these cases, as shown in the table. The overall refinement time (including all components) is equivalent to roughly 24 point-implicit RK3 solver cycles or 1–2 BDF2 time steps (with 25 linear-solver subiterations) for the largest mesh on 6400 cores. These results demonstrate that the current implementation is sufficiently efficient and scalable for use in dynamic WMLES simulations. Although these full-refinement test cases highlight the performance of the CAD-projection operations, they do not assess the transition-element insertion phase, which is addressed separately in Section F.

4. *AMR Interface for Linking CFD solvers*

The AMR API has been implemented using a driver code that calls the AMR library and the flow solver in sequence, using its own set of registered memory pointers to pass information back and forth between these two software applications. This approach was driven by our experience with HELIOS, which uses a Python driver, although the current driver is very lightweight and written in modern Fortran. In general, the information required by the AMR library from the flow solver consists of a single array with one integer per cell, which identifies each cell as either flagged for refinement (ITAG=1), flagged for coarsening (ITAG=-1), or no change (ITAG=0). Conversely, the flow solver receives from the AMR library memory pointers that allow a description of the newly refined and redistributed (i.e., load-balanced) mesh. These include a list of cells and vertices on each partition, along with a classification of vertices as real or ghost, and the required MPI communication schedules (send-receive patterns) for the CFD code. Additionally, the flow solver must supply the current flow solution to the AMR library, which returns the flowfield projected onto the new refined AMR mesh. The Fortran driver code approach has been used to link the AMR library to our in-house NSU3D flow solver[21] as well as the NASA FUN3D code.[22]

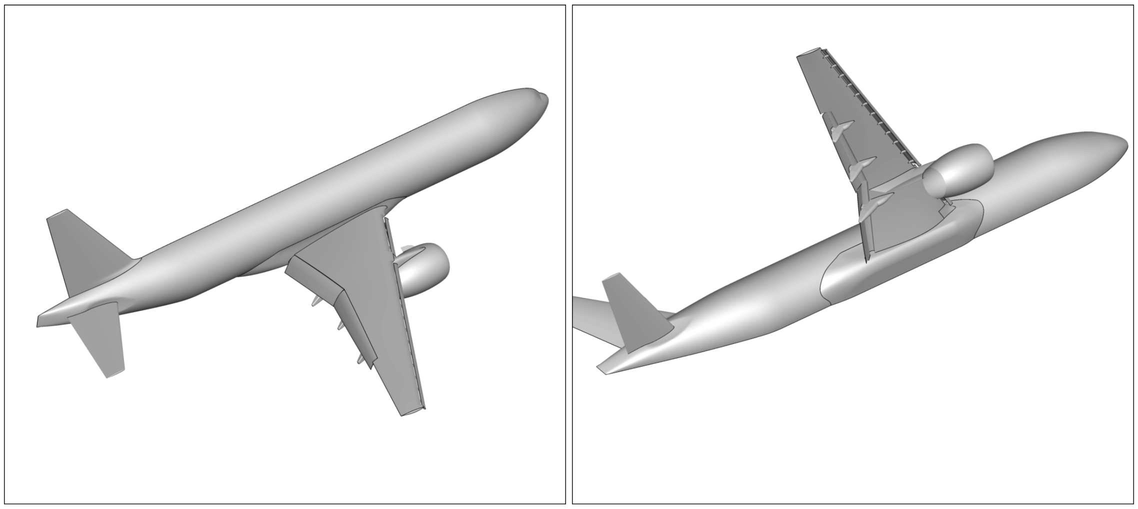
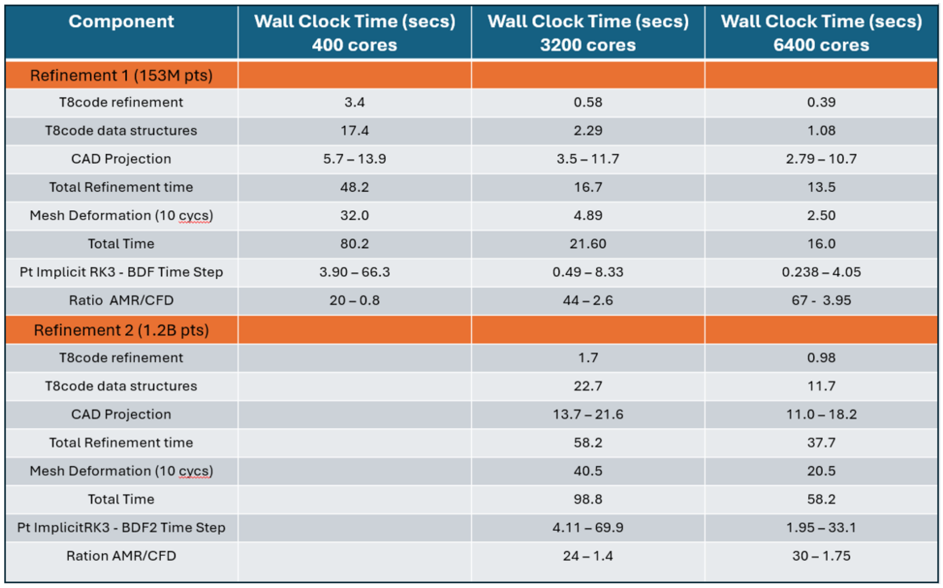


Figure 10. Illustration of HLPW5 Case 2.4 Heldenmesh geometry definition comprising 655 surface patches.

Table 1. Mesh sizes and timings for 2-level refinement of initial coarse Heldenmesh WMLES mesh for HLPW5 Case 2.4 test case running on 6360 cores.



Component	Wall Clock Time (secs) 400 cores	Wall Clock Time (secs) 3200 cores	Wall Clock Time (secs) 6400 cores
Refinement 1 (153M pts)			
T8code refinement	3.4	0.58	0.39
T8code data structures	17.4	2.29	1.08
CAD Projection	5.7 – 13.9	3.5 – 11.7	2.79 – 10.7
Total Refinement time	48.2	16.7	13.5
Mesh Deformation (10 cycs)	32.0	4.89	2.50
Total Time	80.2	21.60	16.0
Pt Implicit RK3 - BDF Time Step	3.90 – 66.3	0.49 – 8.33	0.238 – 4.05
Ratio AMR/CFD	20 – 0.8	44 – 2.6	67 - 3.95
Refinement 2 (1.2B pts)			
T8code refinement		1.7	0.98
T8code data structures		22.7	11.7
CAD Projection		13.7 – 21.6	11.0 – 18.2
Total Refinement time		58.2	37.7
Mesh Deformation (10 cycs)		40.5	20.5
Total Time		98.8	58.2
Pt ImplicitRK3 - BDF2 Time Step		4.11 – 69.9	1.95 – 33.1
Ration AMR/CFD		24 – 1.4	30 – 1.75

II. Test Cases and Results

A. Preliminary Dynamic AMR Demonstration Test Case

The first test case involves WMLES for flow over a sphere using a cell-centered discretization version of our in-house NSU3D flow solver with dynamic AMR. The flow conditions are set as: $Ma = 0.1$, $Re = 4 \times 10^5$. The initial mesh was generated using the GMSH software[23] and contains 50,479 vertices and 260,050 cells. The mesh contains layers of prism elements near the body, while the remainder of the domain is filled with tetrahedral elements. Dynamic AMR is used with four levels of refinements (a total of five mesh levels), and the entire case was run on 240 cores in one batch job. Because this case uses a cell-centered discretization, there are no transition elements in the AMR meshes, and the hanging node interfaces are handled naturally by the CFD discretization as shown in Figure 3. The adaptation criterion is based on the gradient of velocity, and refinements are applied every 50 time steps at the beginning of the simulation, and every 200 time steps for simulation times beyond 3000 time steps, initially using three levels of refinement. After 8000 time steps, one additional refinement using four levels is performed, after which the mesh is frozen, and the simulation is run out on the final adapted mesh, which contains close to 7 million vertices. The AMR is used in a refine-only mode, where cells can be refined in regions of high velocity gradients, but never coarsened. In this manner, the AMR gradually fills in wake regions as they develop. A time history of the number of mesh vertices and cells as the AMR progresses is shown in Figure 11, where the final number of mesh cells is approximately 15 million (with 7 million vertices). Figure 11 shows the drag coefficient history. This figure also provides two snapshots of the dynamic AMR process along with the evolving solution. A full video of the mesh and solution time history is available at https://server.scientific-sims.com/cfdlab/AMR/re4E5.gif

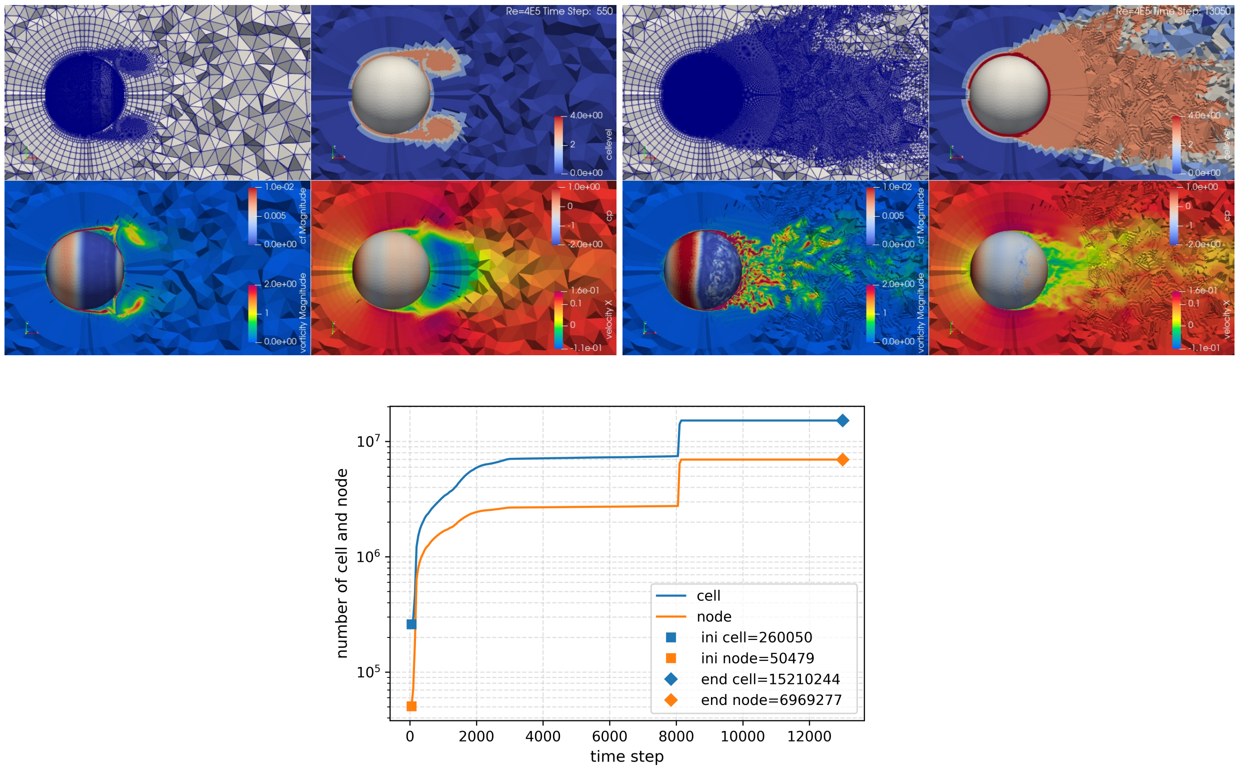


Figure 11. Illustration of dynamic AMR for cell-centered WMLES of flow over sphere; (left) early simulation time, (right) later simulation time. Bottom: Time history of dynamic AMR mesh size in terms of vertices and cells.
Full time history movie file available at: https://server.scientific-sims.com/cfdlab/AMR/re4E5.gif

In these simulations, the new surface points created in the AMR process were linearly interpolated from surrounding mesh points, resulting in a faceted sphere surface, which makes the accurate prediction of drag not feasible for this case. This was one of our first dynamic AMR simulations and served to demonstrate the feasibility of performing dynamic AMR with WMLES simulations in terms of the scalability and efficiency of the AMR process relative to the CFD solution.

B. Steady RANS Demonstration Case

Although the adaptive mesh refinement (AMR) capability is primarily designed for dynamic adaptation in scale-resolving simulations, it can also be applied effectively to steady RANS problems. In such cases, the AMR interface provides an efficient, scalable and streamlined workflow for managing mesh refinement without manual intervention, while still maintaining high grid quality. For this demonstration, the NSU3D solver was used to compute the DLR-F6 wing-body configuration at Mach 0.8, angle of attack $\alpha = 1^\circ$, and Reynolds number $Re = 3 \times 10^6$, based on the mean aerodynamic chord. The flow features a relatively strong transonic shock. This case was computed employing the Roe flux scheme with a limiter, consistent with typical steady RANS practices. The initial unstructured mesh contained approximately 1.2 million points and consisted of tetrahedral elements in the outer regions, with prismatic elements in the boundary layer regions and a small number of pyramidal elements in viscous/inviscid transition regions.

A mesh refinement criterion based on the gradient of density was used. In this case, cells with gradients larger than the mean cell gradient computed over the entire mesh were flagged for refinement at each AMR pass. Only refinement was considered, and no cell coarsening was applied. Although this approach does not constitute an optimal error indicator, it effectively targets shock structures and other regions of strong flow gradients, making it well-suited for demonstration purposes.

Three grid levels were generated: the initial mesh followed by two progressively refined AMR meshes. The solver was run for 500 iterations on each grid level prior to initiating the next AMR pass. Figure 12 shows the three resulting meshes. The first-level mesh contains 1.2 million points, the second 3.9 million points, and the third approximately 14 million points. Refinement occurs in regions of high gradients, including the leading and trailing edges, as well as across the shock waves. The AMR process produces a smooth element distribution and maintains good mesh quality throughout. Figure 13 depicts the computed surface pressure contours on each mesh level, showing increasing suction peaks and finer shock resolution on the adaptively refined mesh levels.

The convergence history of the entire simulation is shown in Figure 14. Convergence on the finest grid exhibits stalling, attributable to limiter chatter in regions of strong shocks. However, the lift coefficient converges rapidly on all three grids, showing a trend towards grid convergence with increasing AMR levels. Furthermore, the smooth changes between grid levels are indicative of the accuracy of the solution interpolation between grid levels, which is nominally second-order accurate. The entire calculation, run on 200 processor cores, required approximately one hour of wall-clock time, dominated by the solution on the finest level, which ran at 6.4 seconds per cycle. The AMR operations accounted for less than 1% of the total runtime due to the large number of flow-solver iterations performed between AMR passes in this steady-state setting. Note that achieving equivalent resolution in the refined regions of the finest AMR mesh (14M points) using a global refinement approach would result in a mesh of 76 million points.

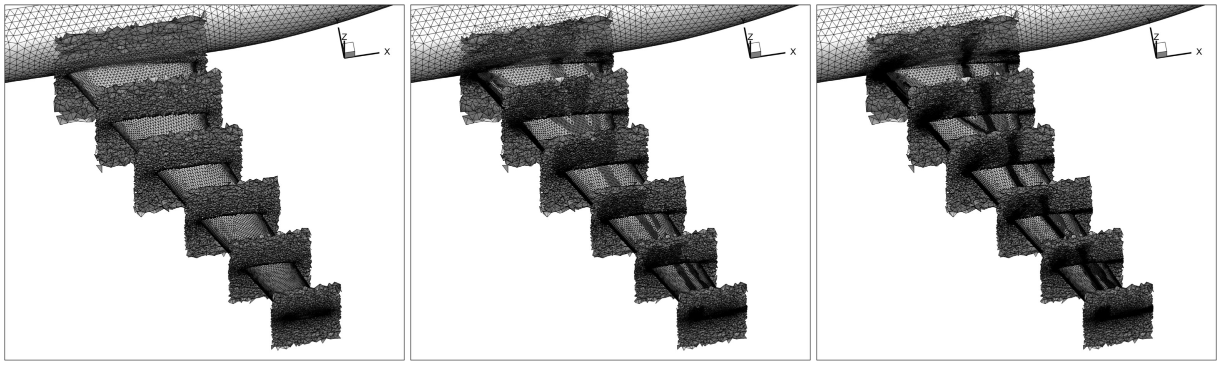


Figure 12. Adaptively refined meshes for NSU3D steady RANS simulation of flow over DLR-F6 wing-body: (left) Initial mesh: 1.2M points, (center) AMR Level 1 mesh: 3.9M pts, (right) AMR Level 2 mesh: 14M pts)

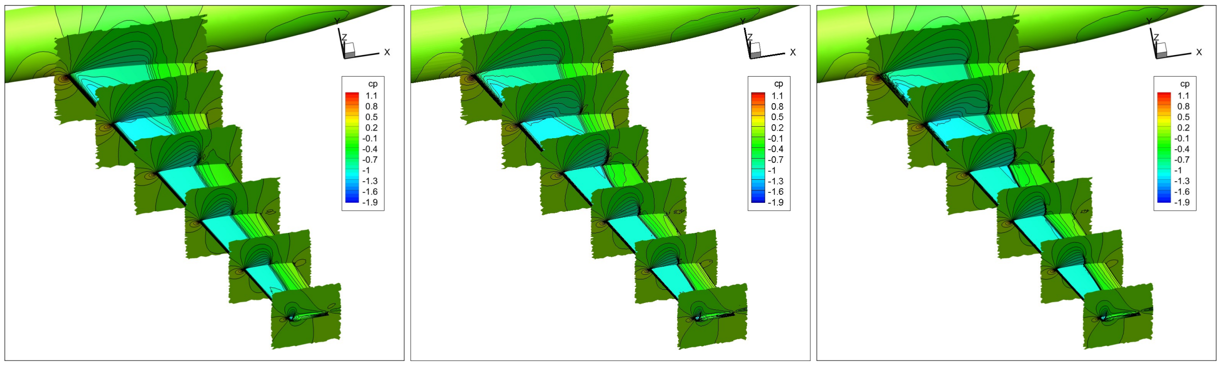


Figure 13. NSU3D computed pressure distribution on adaptively refined meshes for RANS simulation of flow over DLR-F6 wing-body

C. Convecting Vortex Test Case

In order to demonstrate a simple transient case, a convecting vortex test case has been set up and run using the FUN3D-AMR interface. This case is run in URANS mode with FUN3D, using the fully upwind scheme (Roe scheme with $\kappa = 0$) and the BDF2OPT time-stepping scheme. The flow is initialized with an isentropic vortex in a freestream flow with Mach = 0.2 in the x-direction. The domain is periodic in the x-direction, and the simulation is run for two CTUs,

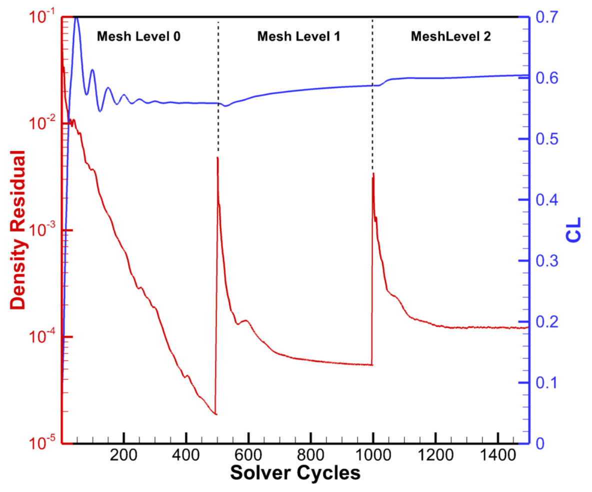


Figure 14. Convergence history for adaptive mesh NSU3D solution of flow over DLR-F6 wing-body

which corresponds to two passes of the vortex through the domain, using 1000 time steps per CTU (time step of 0.1 and total time of 200, domain size 20,1,10 in x-y-z, respectively). The initial mesh contains 14,652 prismatic elements with 11,352 vertices. Ten time steps are performed on the initial mesh, and three refinement levels are added consecutively at time levels 1, 2, and 3 (corresponding to time step numbers 10, 20, and 30. The refinement criterion is based on the density value, where refinement is triggered for density values below 0.99996. Thereafter, the mesh is adapted every 10 time steps, resulting in a refined region that tracks the convecting vortex in space throughout the simulation. Figure 15 depicts the resulting mesh at time levels just prior to the addition of the next refinement level, as well as at further time instances as the vortex convects downstream. The refined AMR mesh varies in time but contains, on average, approximately 260,000 vertices with 11,00 tetrahedra, 12,000 pyramids, and 470,000 prisms. Figure 16 compares the flow solution from the same simulation on the initial (static) mesh with the dynamic AMR mesh, showing much lower diffusion of the vortex structure over time, as expected. The simulation was run on an 8-core laptop computer and required 3.65 seconds per BDF2OPT time step for FUN3D. Each AMR pass required 4.65 seconds, which is equivalent to the time required for 1.27 time steps. Since the AMR passes were only executed every 10 time steps, the AMR time constitutes about 11% of the total simulation time. Although this test case does not involve a turbulence scale-resolving CFD simulation, it demonstrates the effectiveness of the AMR process in preserving flow features with localized refinement and coarsening, and will prove useful for cases with isolated larger-scale flow features such as tip vortices or strake wakes.

D. Channel Flow Test Case

The LES community has considerable experience with channel-flow simulations, making this configuration an ideal initial test case for turbulence-resolving methods. Furthermore, the characteristics of an "optimal" grid are reasonably well established, providing a good benchmark for evaluating AMR for scale-resolving simulations. For this case, the first task consists of establishing a well-resolved baseline solution with FUN3D to be used as a reference for comparing the accuracy of subsequent AMR solutions. A relatively fine mesh consisting of purely prismatic elements with 38,093,825 vertices was constructed, matching the resolution described in reference.[24] Periodic boundary conditions are applied in the streamwise and spanwise directions. The bulk Mach number for this case is 0.2, and the conditions are identical to those given in reference.[24] A non-dimensional streamwise pressure gradient value of $dpdx = 1.148E-4$ and the value of $\kappa = 0.9$ were specified, where the κ parameter determines the level of blending between upwind and central difference schemes in FUN3D for reduced dissipation with higher κ values. The flow was initialized using a mean channel-flow velocity profile with random perturbations of amplitude 45% of the freestream velocity. In all cases, the flow solution was run out to a minimum nondimensional time of 45, where the time variable is nondimensionalized by the streamwise length of the computational domain over the bulk velocity L_x/U_b.

Two AMR runs were also performed using an isotropic version of the filtered velocity refinement criterion de-

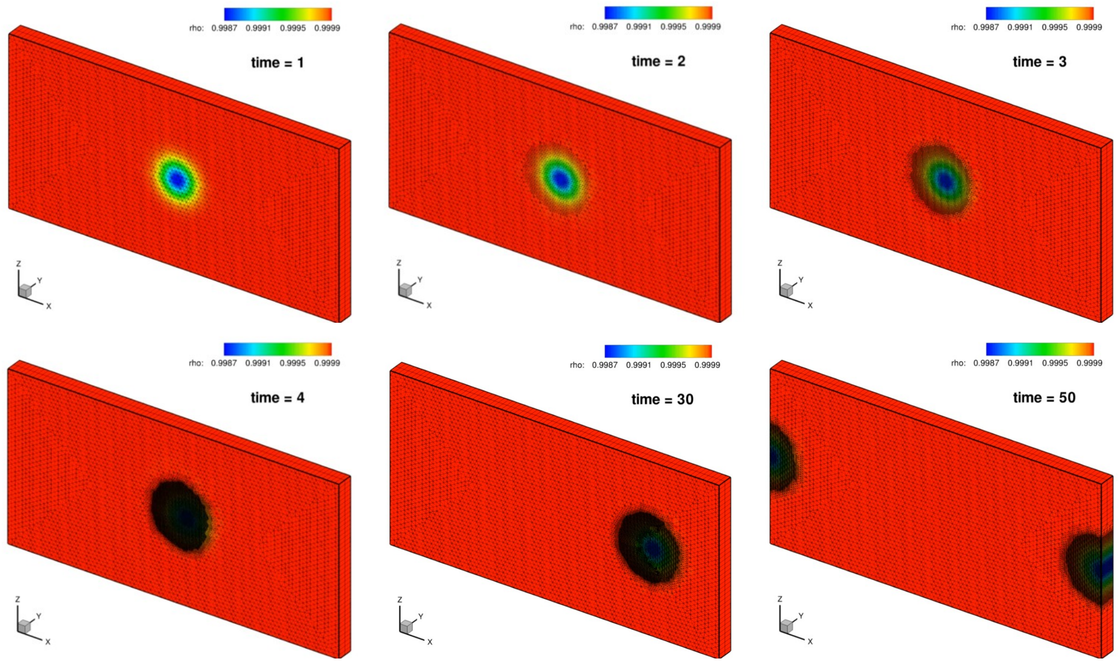


Figure 15. Illustration of the AMR grid used for the convecting vortex test case. Initial mesh at first time step is refined around vortex region by three levels at time levels $t = 2$, $t = 3$, and $t = 4$, respectively, using a time step size of $\Delta t = 0.1$, and the refined region follows the convecting vortex thereafter by performing an AMR pass every 10 time steps.

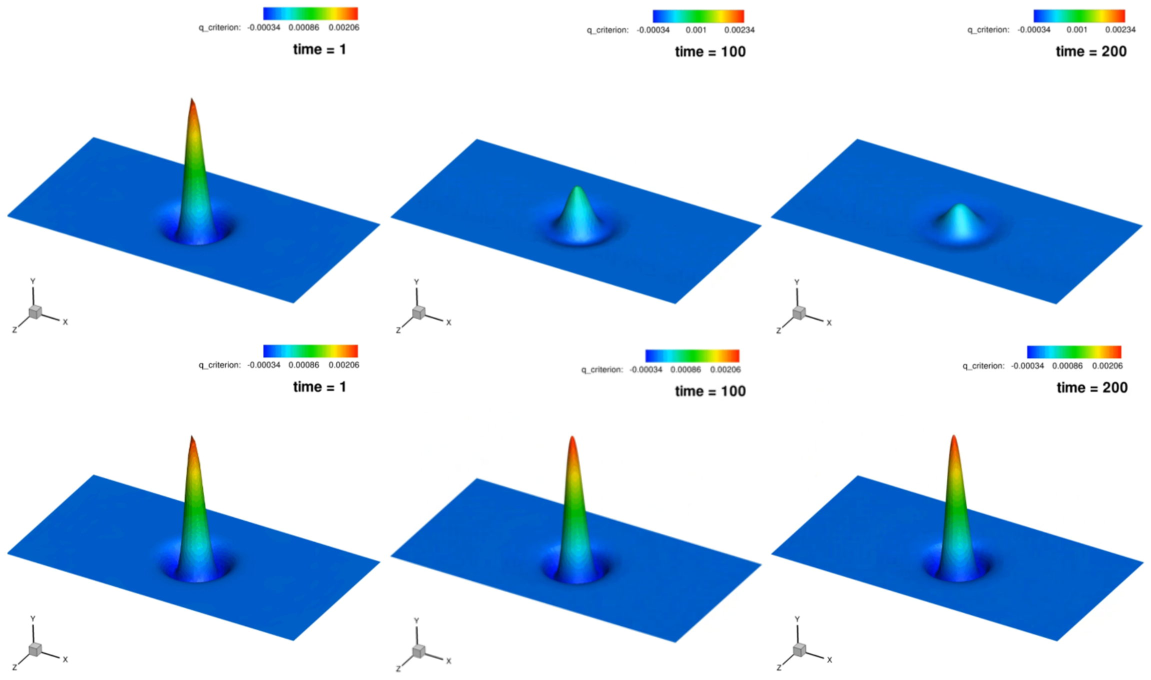

q_criterion: -0.00034 0.00086 0.00206
time = 1
q_criterion: -0.00034 0.001 0.00234
time = 100
q_criterion: -0.00034 0.001 0.00234
time = 200
q_criterion: -0.00034 0.00086 0.00206
time = 1
q_criterion: -0.00034 0.00086 0.00206
time = 100
q_criterion: -0.00034 0.00086 0.00206
time = 200
Y
X
Z

Figure 16. Illustration of convecting vortex at initial time and after one and two passes through the periodic mesh domain. Upper row: Static mesh, Lower row: AMR mesh. Results obtained using the AMR interface linked to FUN3D.

scribed in reference.[25] The essence of this criterion is to provide a measure of the energy of the small turbulence scales to drive the AMR process. The smallest resolved scales can be extracted as

$$u^* = u' - \hat{u} \tag{1}$$

where u' is the instantaneous velocity field and $\hat{u}$ denotes a spatially low-pass filtered velocity. In this work, we construct an isotropic filtered velocity field as:

$$u' = \hat{u} - \alpha^2 \frac{\partial^2 \hat{u}}{\partial x_i^2} \tag{2}$$

where α defines the filter width and is taken as $\alpha^2 = \frac{\Delta^2}{4}$, and $\Delta = Vol^{\frac{1}{3}}$ is a measure of the local mesh element size. Equation (2) corresponds to an implicit smoothing equation of the instantaneous velocity field, which is discretized by Galerkin finite elements and solved for $\hat{u}$ to machine precision using O(100) Jacobi iterations. The error indicator at a point is then defined as

$$A = \sqrt{\langle u_i^* \cdot u_i^* \rangle} \tag{3}$$

where the subscript i denotes the x-y-z velocity components. The error indicator A is averaged in time between refinement passes as

$$A = \frac{1}{n} \sum_{k=1}^{n} (A_i) \tag{4}$$

where n is the number of samples used for the temporal averaging. With a user chosen threshold value A_{thresh}, the target grid spacing is defined as

$$\Delta_{target} = \left(\frac{A_{thresh}}{A} \right)^{1/\beta} \Delta \tag{5}$$

or

$$r_\Delta = \frac{\Delta_{target}}{\Delta} = \left(\frac{A_{thresh}}{A} \right)^{1/\beta} \tag{6}$$

In this case, the value $\beta = 2$ has been used, corresponding to a second-order accurate spatial discretization. Finally, a vertex-based refinement tag is constructed as:

$$itag = \begin{cases} 1 & r_\Delta < r_{\Delta Lower} & \text{refinement} \\ 0 & r_{\Delta Lower} \leq r_\Delta \leq r_{\Delta Upper} & \text{no change} \\ -1 & r_\Delta > r_{\Delta Upper} & \text{coarsening} \end{cases} \tag{7}$$

This refinement tag is averaged to the cells and passed to the AMR library in order to generate the next AMR mesh.

In practice, this approach was found to generate sporadic regions of refinement in the domain due to the random turbulent fluctuations. Although these diminish with increased time averaging, a spatial averaging process was also applied to the error indicator A in order to obtain smoother refinement patterns. An anisotropic version of the low-pass filter described above is used as the space smoothing operator for the error indicator A:

$$A = \hat{A} - \frac{\partial}{\partial x_i} \left(p_{ij} \frac{\partial \hat{A}}{\partial x_j} \right) \tag{8}$$

where p_{ij} is a metric tensor. For smoothing only in the x, y and z directions, the metric tensor can be defined as

$$p_{ij} = \begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & c_z \end{bmatrix} \tag{9}$$

By adjusting the values of the matrix p_{ij}, smoothing can be achieved in any specific direction on the unstructured mesh. For the channel flow test case, smoothing has been applied in the spanwise and streamwise directions by setting $c_x \approx c_y \gg c_z$.

The initial prismatic mesh for the AMR simulations contained 79,233 vertices, obtained as a uniform coarsening of the fine mesh described above. Three levels of refinement (i.e., four mesh levels including the initial mesh) were used, and refinement passes were performed every 10,000 time steps. The refinement criterion A was evaluated every 20 time steps, generating 500 samples for time averaging between refinement passes. Two AMR simulations were performed, denoted as AMR1 and AMR2, using the different threshold values $A_{thresh} = 0.004$ and $A_{thresh} = 0.002$, respectively. The meshes of AMR1 and AMR2 were frozen after 12 and 17 refinement passes(120,000 and 170,000 time steps), respectively. Then, the solution statistics were gathered over a subsequent period of 100,000 time steps.

Figure 17 illustrates the computed solutions at the end of the simulation for the static fine mesh and AMR meshes, which are qualitatively similar. A perspective view of the computational domain and corresponding final meshes is shown in Figure 18. The drag coefficient histories for all three simulations (static fine mesh and two AMR simulations) are shown in Figure 19, showing good agreement with the reference value of 0.0058 and in agreement with results in reference.[24] The second part of the figure shows the growth history in the mesh size for the AMR solutions compared with the static fine mesh resolution. The final meshes for the two AMR solutions contain 6.9 and 12.7 million points for threshold levels of $A_{thresh} = 0.004$ and $A_{thresh} = 0.002$, representing a factor of 6 and 3 reduction compared to the fine mesh, respectively. Figures 20 through 23 illustrate the physical mesh size distribution along with the mesh spacing as a function of the wall-normal distance z and z^+, for the fine mesh, as well as the initial and final AMR meshes. The AMR meshes show significant refinement in the near-wall region, with much coarser final resolution in the center of the channel compared to the static fine mesh. Comparing Figure 22 and 23 shows that the AMR1 solution maintains the initial coarse mesh resolution in the center of the channel, whereas the lower threshold AMR2 solution applies one level of refinement in this region. Perhaps more importantly, the AMR2 solution also results in additional refinement in the log layer $z^+ \approx 100$ region compared to the AMR1 solution.

Figure 24 depicts the mean velocity profiles computed for all three simulations, which compare well with the results from reference[24] and the log layer profile near the wall. The figure also compares the computed Reynolds stress components for all three simulations with those from reference.[24] Good agreement is observed for the cross terms, although the $\overline{u'u'}$ term is slightly overpredicted by the AMR simulations when compared to the equivalent results in the reference. However, the AMR2 results, which resulted in finer wall resolution, compares more favorably with the current fine mesh and reference results.

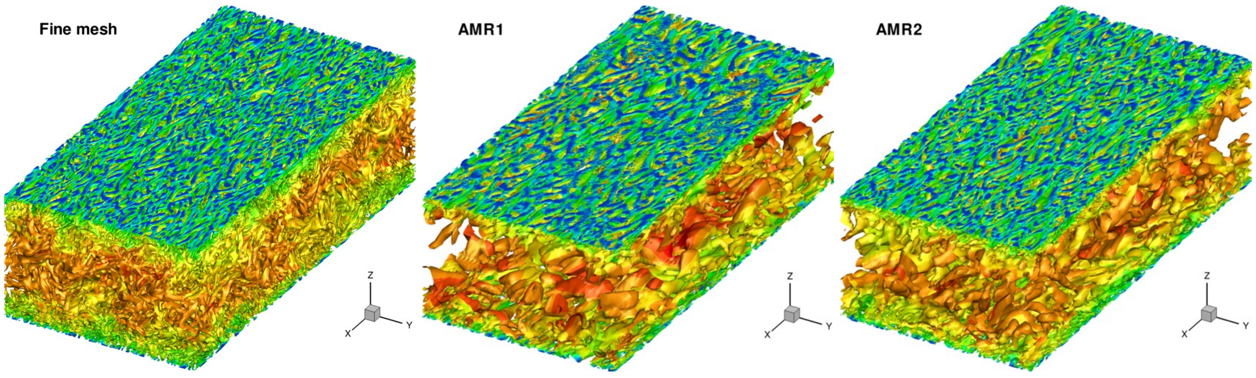


Figure 17. Illustration of turbulent channel flow computed with FUN3D on fine mesh (left) and on AMR meshes (right)

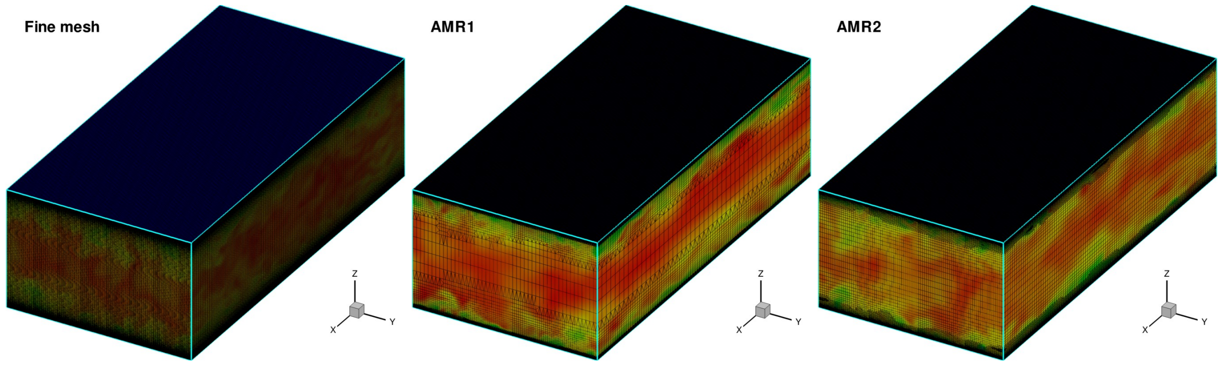


Figure 18. Fine mesh (38M pts) and adaptively refined meshes for Channel Flow test case.

E. Backwards Facing Step Test Case

The backward-facing step test case was simulated using FUN3D linked to our AMR interface. This test case has been computed previously by Toosi and Larsson,[25] which will serve as a basis for comparison in this work. The step height is taken as 1.0 and is located at the origin of the streamwise x coordinate system. The Reynolds number based on the step

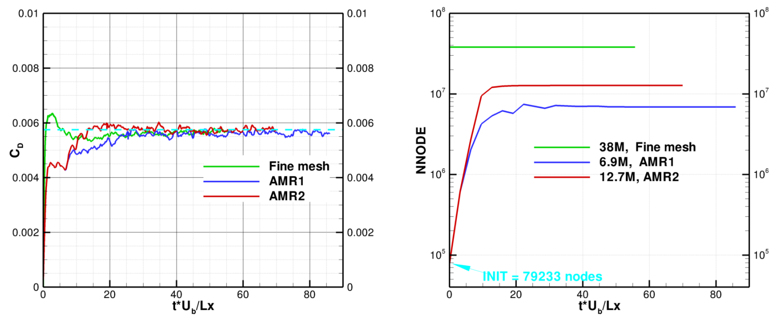


Figure 19. (left) Drag coefficient history for channel flow simulations; (right) History of dynamic AMR mesh sizes.

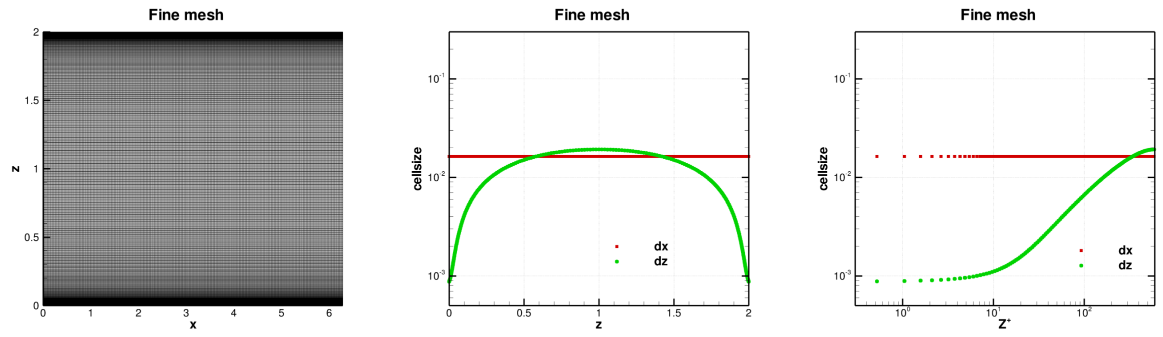


Figure 20. Spanwise cut of fine grid and fine grid resolution as a function of z (wall normal direction) and z^+ (over half channel).

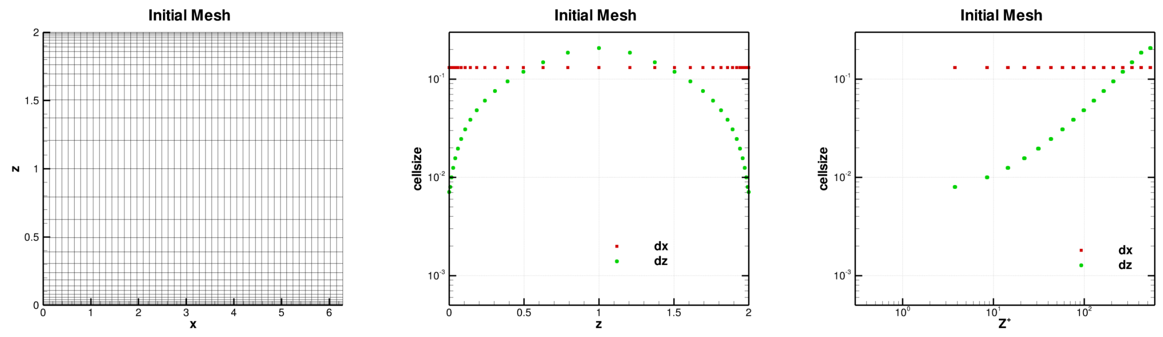


Figure 21. Spanwise cut of initial AMR grid and initial grid resolution as a function of z (wall normal direction) and z^+ (over half channel).

height h and inflow velocity U_{inf} is $Re_h = 5100$, the inlet Mach number is 0.2, and the top boundary is a slip wall placed 5h above the upstream wall. The computational domain extends 20h upstream of the step, 25h downstream of the step, and has a width of 4h. Periodic boundary conditions are used in the spanwise direction. This flow contains an attached turbulent boundary layer upstream of the step, a free shear layer after the separation, an impingement/reattachment region, and a large recirculation zone. The combination of multiple different building-block flow types makes this a good test case to assess the AMR capability. A small step of height 0.4 is placed at the base of the inlet boundary in order to cause separation-induced turbulence in the inlet boundary layer. The value $\kappa = 0.9$ was also used in the FUN3D simulations, since lower values were found to suppress turbulence ahead of the step. This combination of inlet treatment and κ value resulted in a turbulent boundary layer ahead of the backwards-facing step as seen in the solution shown in Figure 25.

The solution was run for a total of 260 CTUs, using 1000 BDF2OPT time steps per CTU (based on the step height).

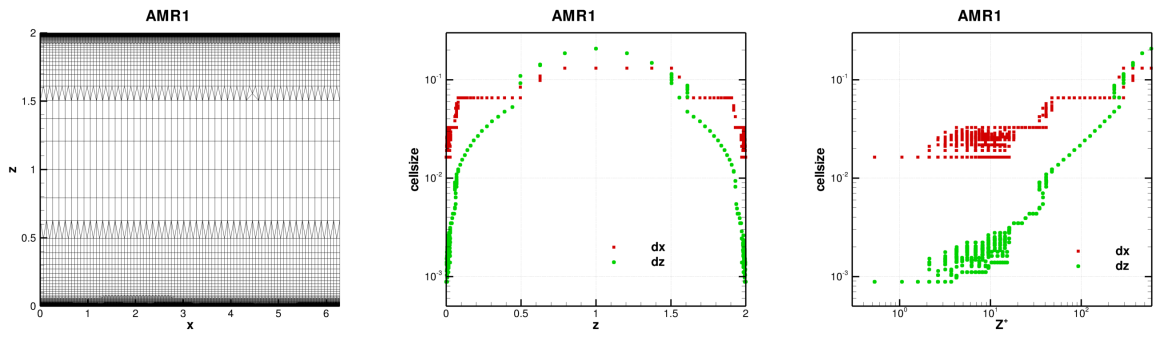


Figure 22. Spanwise cut of final AMR1 grid and AMR1 grid resolution as a function of z (wall normal direction) and z^+ (over half channel).

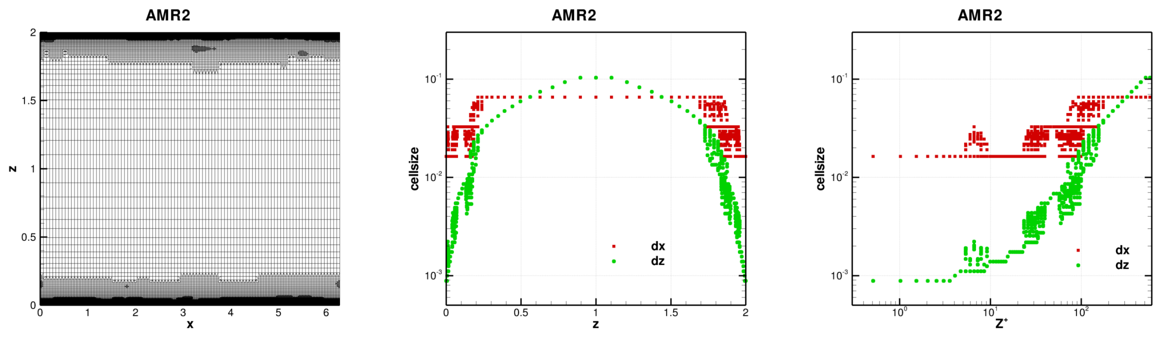


Figure 23. Spanwise cut of final AMR2 grid and AMR2 grid resolution as a function of z (wall normal direction) and z^+ (over half channel).

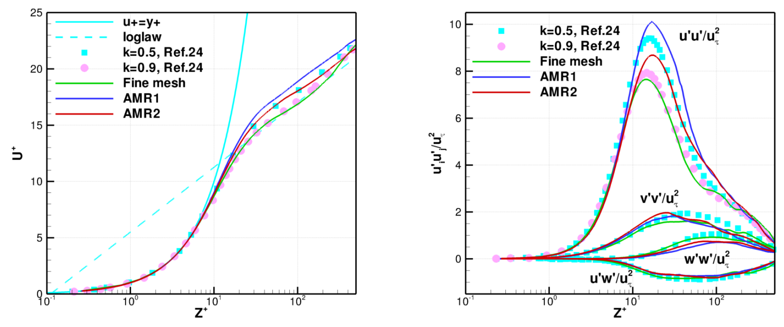


Figure 24. (left) Computed mean velocity profiles for fine grid and AMR grid simulations compared with results from reference[24] and log layer profile; (right) Computed Reynolds stress terms compared with results from reference.[24]

The initial mesh contained 195,300 vertices, and AMR passes were performed every 5000 time steps over 210,000 time steps (total of 42 AMR passes) using the filtered velocity refinement criterion from reference.[25] In this case, the value $A_{thresh} = 0.002$ was used and the A indicator was sampled every 20 time steps. The AMR was subsequently frozen and an additional 50 CTUs or 50,000 time steps were performed for extracting flow statistics. The final mesh contained a total of 13,671,435 vertices, using three levels of refinement. Figure 25 illustrates the adapted mesh at 69 CTUs showing refinement along the turbulent boundary layer upstream of the step, as well as in the shear layer region downstream of the step. The reattachment line is seen to be approximately at the $x = 5$ location, which is slightly upstream of the experimental and computational results reported by Toosi and Larsson.[25] The refined meshes are seen to capture the shear layer that develops over the top of the step, and also focus on the reattachment region, while leaving the recirculation region near the base of the step relatively coarse. These mesh characteristics are consistent with those described in reference.[25]

Figure 26 compares the mean velocity and Reynolds stress terms at two different streamwise locations with the DNS results reported in[25] and with accompanying experimental results from reference.[26]

Overall, the results show reasonable agreement. The mean velocity and the streamwise normal Reynolds stress component are overpredicted by the AMR simulation. However, this effect is less pronounced in the downstream $x = 10$ location, suggesting that the discrepancies may be due to the inlet boundary layer turbulence characteristics. This may not be surprising given the simple approach used to trip the inlet boundary layer. Work is underway to look at more sophisticated inlet profile specification methods and to quantify the inlet boundary layer profile for both static mesh and AMR solutions.

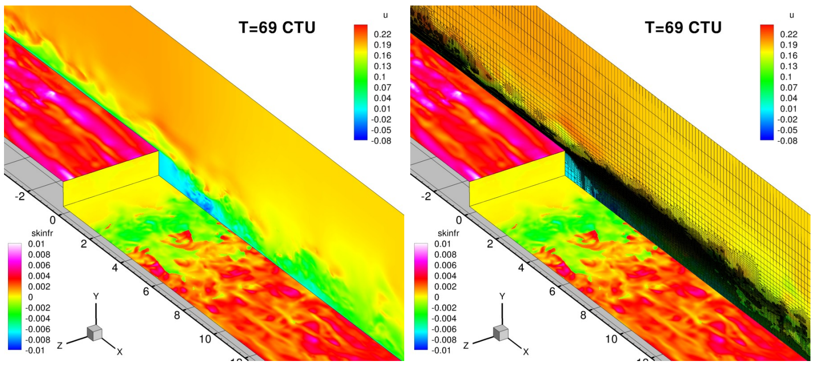


Figure 25. Instantaneous velocity and skin friction profiles and adaptive mesh for AMR simulation of backwards facing step at time of 69 CTUs based on step height.

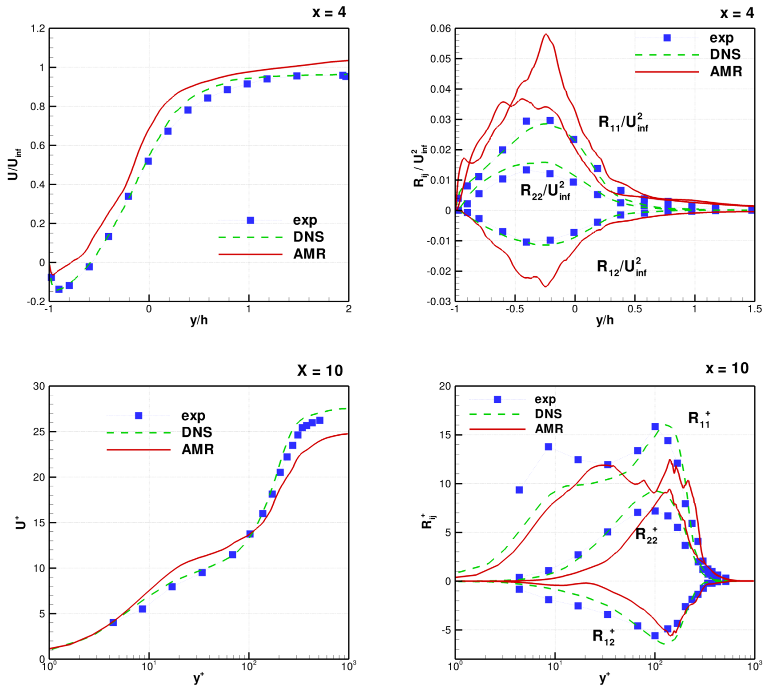


Figure 26. Computed mean velocity and Reynolds stress profiles at $x = 4$ and $x = 10$ streamwise stations compared with DNS results from reference[25] and experimental results from reference.[26]

F. High-Lift Test Case

In this section, we present WMLES results using dynamic AMR for the HLPW5 Case 1 wing–body configuration,[20] computed with the NSU3D flow solver. At the time these simulations were performed, the CAD-based surface point-projection capability and WMLES-specific refinement criteria were not yet operational, and a simple density-gradient refinement criterion was employed in the AMR process. As a result, the outcomes shown here are primarily qualitative and are intended to demonstrate the feasibility of this approach for WMLES of industrially relevant configurations.

For the current study, a coarse mesh containing 5.4 million points and 14 million cells (5 million tetrahedra, 2 million pyramids, and 7 million prisms) was provided by Helden Aerospace Inc. This mesh was designed to be approximately 2x coarser in each spatial direction than the XC mesh for WMLES posted by Helden Aerospace on the HLPW5 web site, allowing for multiple adaptive refinements. The original workshop XC mesh was designed to have a first cell spacing of 0.21 inches and approximately $y^+ = 160$, whereas the current mesh has a normal spacing of approximately 0.42 inches, corresponding to roughly $y^+ = 320$.

In a preliminary study, we performed a single refinement level of this initial mesh to test the capability at scale and to examine the quality of the resulting refined mesh regions, including the use of transition elements for NSU3D in WMLES mode. Figure 27 depicts the original and refined surface mesh in the region of the leading edge of the wing, while Figure 28 depicts the two meshes in the region of the fuselage-symmetry plane intersection, where the quadrilateral faces of the prism elements are visible on the symmetry plane. In both regions, the refined mesh region is seen to reproduce smaller cells of similar aspect ratio and quality to those of the original coarse mesh, as expected. Furthermore, the transition elements can be seen to provide a smooth change from the refined regions to the coarser regions, both on the wing-body surface and through the boundary layer as seen on the symmetry plane.

The next step involved running the WMLES simulation for HLPW5 Case 1 with a sequence of adaptively refined meshes. The flow conditions for HLPW5 Case 1 are Mach = 0.2, Incidence = 11^o, and Reynolds = 5.6 million. We initiated the runs on 6360 cores in order to test the scalability of the entire AMR-CFD procedure, as well as to enable timely simulation on the resulting refined meshes. We ran 3 refinements throughout the course of the simulation, with the mesh increasing from 5.4 million points to 44 million points on the first refinement pass, to 220 million points on the second refinement pass, and to 1.02 billion points on the third refinement pass. NSU3D was run on the first (original), second, and third meshes of the refinement sequence. A solution on the fourth mesh was not attempted due to resource limitations, but the mesh was generated during the same simulation run in order to demonstrate the efficiency and scalability of the AMR process.

Figure 29 depicts the computed iso-surfaces of Q-criterion, colored by vorticity magnitude for the first three meshes of the sequence. The figure shows the capture of finer turbulence scales in the WMLES simulation as the mesh is adaptively refined, with the finest scales observed on the third mesh of the sequence, especially near the wing root and upper surface of the fuselage.

Table 2 provides a summary of the adaptively generated meshes in this WMLES calculation and the wall-clock time required in the different phases of the AMR process. As mentioned previously, this case does not include the CAD surface point projection, but includes the insertion of transition elements. On the last refinement pass, the mesh resolution is increased from 220 million points (519 million cells) to 1.02 billion points (2.25 billion cells), which requires a total wall-clock time of 48 seconds, running on 6360 cores. This is broken down into 9.9 secs for the t8code element subdivision and load rebalancing operations, 11.3 secs for the rebuilding of data structures, and 27.6 secs for the insertion of transition elements. This corresponds to approximately the time requried for 1.7 BDF2 time steps of the finest 1.02 billion point mesh running on 6360 cores.

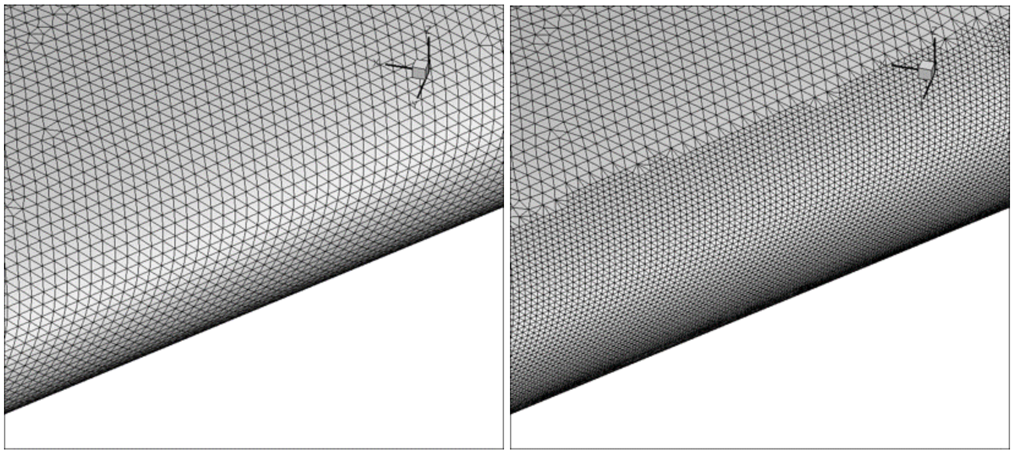


Figure 27. Initial (left) and first level adaptively refined (right) mesh generated on HLPW5 Case 1 HLCRM-WB in leading edge region illustrating preservation of mesh quality and cell aspect ratio in AMR process.

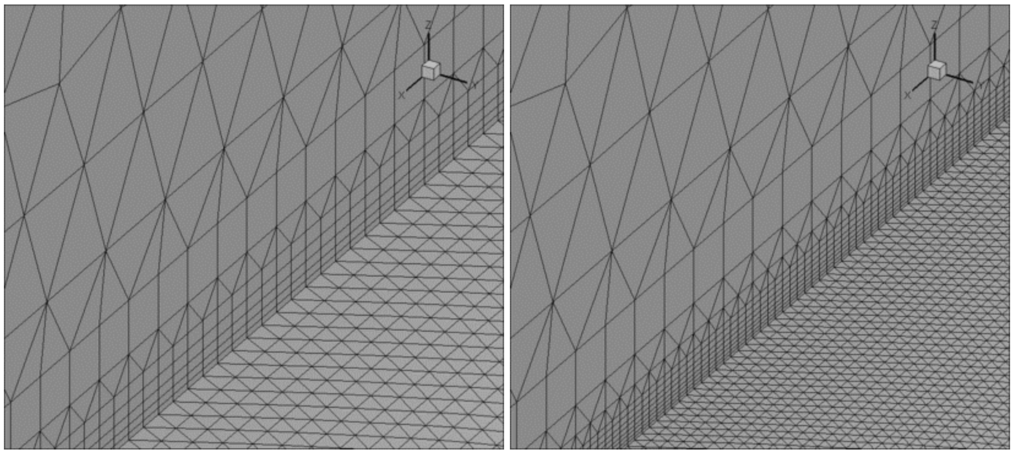


Figure 28. Initial (left) and first level adaptively refined (right) mesh generated on HLPW5 Case 1 HLCRM-WB in fuselage-symmetry plane junction region illustrating preservation of mesh quality and cell aspect ratio in AMR process.

Table 2. Mesh sizes and timings for 3-level adaptive refinement of initial coarse Heldenmesh WMLES mesh for HLPW5 Case 1 test case running on 6360 cores.

	Points	Cells	T8code Library	Data Structures	Transition Elements	Total AMR Time
Mesh 1	5.6M	14M	--	--	--	--
Mesh 2	44M	117M	0.2 secs	0.18 secs	1.0 secs	1.4 secs
Mesh 3	220M	519M	0.8 secs	1.63 secs	3.0 secs	5.5 secs
Mesh 4	1.02B	2.25B	9.9 secs	11.3 secs	27.6 secs	48 secs

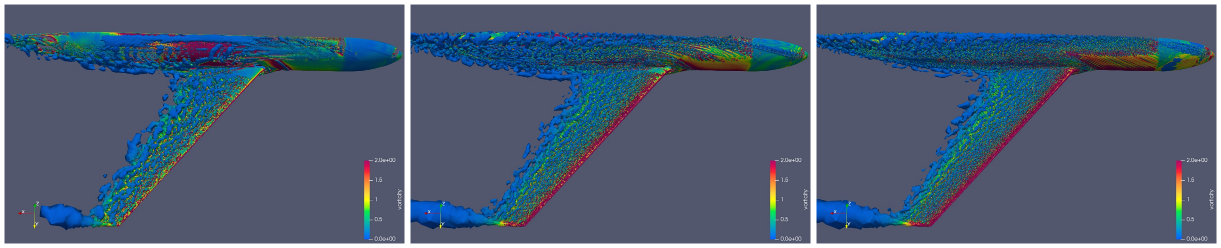


Figure 29. NSU3D WMLES isosurfaces of Q criterion colored by vorticity magnitude showing increased small scale resolution on AMR meshes; (left) original Heldenmesh 5M pts, (center) first level AMR, (right) second level AMR.

Figure 30 shows the time-histories of the force coefficients computed on the first and second refined meshes of the sequence, compared to NSU3D results obtained previously for the workshop running on the XC and C (static) workshop provided meshes, for reference. (The initial mesh of the AMR sequence, with only 5.4M points, is considered too coarse, and results are not shown for this mesh.) In these simulations, the flowfield was reinitialized to freestream after each adaptive mesh refinement operation, before resuming the WMLES flow solution process on each level. Although this approach is not optimal from an efficiency standpoint, it avoids any issues of solution hysteresis for this initial investigation.

The WMLES solution on each mesh level was run for 24 CTUs with a time step of 0.001, and from 24 to 36 CTUs using a smaller time step of 0.0005. The results for the first AMR mesh level are close to those of the XC workshop mesh, while the second AMR mesh results fall in between those of the XC and C workshop mesh levels, indicating correct trends with increasing refinement.

It should be noted that the wall normal resolution of the AMR meshes is significantly smaller than that of the static (XC or C) meshes against which they are compared in this plot. For the second refined mesh, the wall normal resolution is in the range of $y^+ = 80$, which led to the third refined mesh (not shown here and not used for a WMLES solution) having a wall normal resolution of $y^+ = 40$, which is too low for WMLES runs. A floor on the refinement level near the wall can be used to avoid over-resolving this region of the boundary layer for WMLES, as shown in reference,[2,14] although this has not been attempted here.

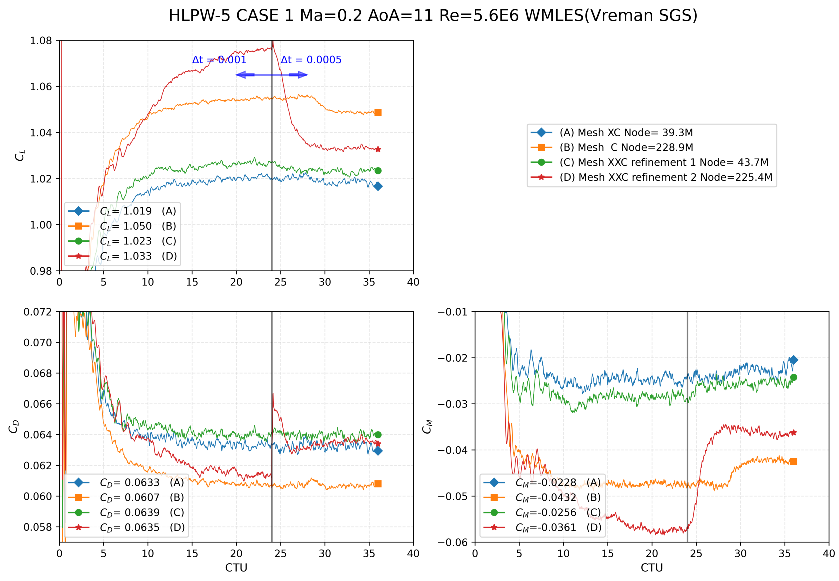


Figure 30. NSU3D WMLES force time histories for HLPW5 Case 1 on adaptively refined meshes compared to original Heldenmesh generated XC and C workshop meshes.

III. Conclusions and Future Work

In this work, we have demonstrated an AMR strategy based on isotropic mesh-element subdivision for both RANS and WMLES CFD flow solvers. Although isotropic subdivision does not generally yield meshes as optimal as those produced by anisotropic remeshing, the method is very efficient and highly scalable, and our interface is designed to integrate tightly into existing workflows for legacy CFD solvers. This has been illustrated here using both the NSU3D and FUN3D flow solvers.

The approach has been applied to steady and unsteady RANS cases as well as WMLES problems, spanning canonical benchmarks through configurations of industrial complexity. Ongoing work focuses on further development and validation of refinement criteria for WMLES, and on quantifying the accuracy and efficiency benefits of the AMR procedure for both RANS and WMLES problems.

A second major thrust of this work is the extension of the AMR interface to flow solvers running on GPU architectures. Our initial strategy involves coupling multi-GPU flow solvers to the AMR interface running on all available CPU cores. In a subsequent phase, AMR operations will be progressively migrated to GPU kernels, although the t8code library will remain CPU-resident, as this is a third-party library and its performance appears to be fast enough to remain CPU-resident without creating a workflow bottleneck.

IV. Acknowledgments

This work was supported by the National Aeronautics and Space Administration (NASA) under a Phase II Small Business Innovation Research (SBIR) award (Contract Number 0NSSC24CA076). The authors appreciate NASA support and guidance throughout the project. We extend our appreciation to the NASA technical monitor, P. Balakumar, as well as to Bill Jones, Li Wang and other NASA subject-matter experts whose feedback and discussions contributed substantially to the development of this effort.

We are also deeply grateful for the contributions and expertise of Johan Larsson and the members of WHOOSH HPC Labs in formulating and implementing the LES refinement crietrion and their assistance with validation test cases. In addition, we acknowledge the valuable contributions and advice of Andrew Wick and Rick Hooker from Helden Aerospace Inc., whose efforts were instrumental in enabling the CAD surface point-projection technology described in this work.

References

[1]DLR-AMR, "DLR-AMR/T8CODE: Parallel algorithms and data structures for tree-based AMR with arbitrary element shapes." https://github.com/DLR-AMR/t8code.

[2]Blind, M. P., Kahraman, A. B., Larsson, J., and Beck, A., "Residual estimation for grid modification in wall-modeled large eddy simulation using unstructured high-order methods," *Computers & Fluids*, Vol. 254, 2023, pp. 105796.

[3]Wissink, A., Jayaraman, B., Datta, A., Sitaraman, J., Potsdam, M., Kamkar, S., Mavriplis, D. J., Yang, Z., Jain, R., Lim, J., and Strawn, R., "Capability Enhancements in Version 3 of the Helios High Fidelity Rotorcraft Simulation Code," *50th AIAA Aerospace Sciences Meeting and Exhibit*, January 09–12 2012, AIAA Paper 2012–713.

[4]Wissink, A. M., Hornung, R. D., Kohn, S. R., Smith, S. S., and Elliott, N., "Large scale parallel structured AMR calculations using the SAMRAI framework," *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 2001, pp. 6–6.

[5]Burstedde, C., Wilcox, L. C., and Ghattas, O., "p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forest of Octrees," *SIAM Journal on Scientific Computing*, Vol. 33, No. 3, 2011, pp. 1103–1133.

[6]Brazell, M. J., Kirby, A. C., and Mavriplis, D. J., "A high-order discontinuous-Galerkin octree-based AMR solver for overset simulations," *23rd AIAA Computational Fluid Dynamics Conference*, 2017, AIAA Paper 2017–3944.

[7]Yoon, S. H., Kirby, A., and Mavriplis, D., "Pseudo-time Stepping Strategies for Space-Time Discontinuous Galerkin Discretizations," *AIAA Scitech Meeting, National Harbor, MD*, January 2023, AIAA Paper 2023–0662.

[8]Holke, J., *Scalable algorithms for parallel tree-based adaptive mesh refinement with general element types*, Ph.D. thesis, University of Bonn, 2018.

[9]Holke, J., Markert, J., Knapp, D., Dreyer, L., Elsweijer, S., Böing, N., Lilikakis, I., Fussbroich, J., Leistikow, T., Becker, F., et al., "t8code - modular adaptive mesh refinement in the exascale era," *Journal of Open Source Software*, 2025.

[10]Holke, J., Knapp, D., and Burstedde, C., "An optimized, parallel computation of the ghost layer for adaptive hybrid forest meshes," *SIAM Journal on Scientific Computing*, Vol. 43, No. 6, 2021, pp. C359–C385.

[11]Holke, J., "Exascale-ready adaptive mesh refinement and applications in Earth system modelling," *19th Workshop on high performance computing in meteorology*, 2021.

[12]Kavouklis, C. and Kallinderis, Y., "Parallel adaptation of general three-dimensional hybrid meshes," *Journal of Computational Physics*, Vol. 229, No. 24, 2010, pp. 8912–8934.

[13]Jansen, K. E., Rasquin, M., Evans, J. A., Brown, J., Carothers, C., Shephard, M. S., Sahni, O., Smith, C. W., Fang, J., Bolotnov, I., Williams, T., and Balakrishnan, R., "Extreme Scale Unstructured Adaptive CFD: From Multi-phase Flow to Aerodynamic Flow Control," ALCF ESP Technical Report ANL/ALCF/ESP-17/8, Argonne Leadership Computing Facility, Argonne National Laboratory, September 2017.

[14]Park, M., "HLPW-4/GMGW-3: Mesh Adaptation for RANS Technology Focus Group Workshop Summary," *AIAA Aviation Meeting, Chicago, IL*, June 2022, AIAA Paper 2022–3210.

[15]Wood, S. and Park, M., "FUN3D Mesh Adaptation Simulations of 4th AIAA High-Lift PredictionWorkshop Case 1a," *AIAA Aviation Meeting, Chicago, IL*, June 2022, AIAA Paper 2022–3394.

[16]Balan, A., Park, M. A., Anderson, W. K., Kamenetskiy, D. S., Krakos, J. A., Michal, T., and Alauzet, F., "Verification of anisotropic mesh adaptation for turbulent simulations over ONERA M6 wing," *AIAA Journal*, Vol. 58, No. 4, 2020, pp. 1550–1565.

[17]Alauzet, F., Frey, P., George, P., and Mohammadi, B., "3D transient fixed point mesh adaptation for time-dependent problems: Application to CFD simulations," *Journal of Computational Physics*, Vol. 222, No. 2, 2007, pp. 592–623.

[18]Mavriplis, D., "Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes," *International journal for numerical methods in fluids*, Vol. 34, No. 2, 2000, pp. 93–111.

[19]Yang, Z. and Mavriplis, D. J., "A Mesh Deformation Strategy Optimized by the Adjoint Method on Unstructured Meshes," *AIAA Journal*, Vol. 45, No. 12, 2007, pp. 2885–2896, doi:10.2514/1.30592.

[20]Kiris, C. C., Ghate, A., and Angel, J., "High Lift PredictionWorkshop 5: Wall-Modeled Large Eddy Simulations Technology Focus Group Summary," *AIAA SCITECH 2025 Forum*, January 2025, AIAA Paper 2025–0049.

[21]Mavriplis, D. J. and Mani, K., "Unstructured Mesh Solution Techniques using the NSU3D Solver," AIAA Paper 2014-081, 52nd Aerospace Sciences Meeting, National Harbor, MD.

[22]Anderson, W. K., Biedron, R. T., Carlson, J., Derlaga, J. M., Diskin, B., Druyor, C. T. J., Gnoffo, P. A., Hammond, D. P., Jacobson, K. E., Jones, W. T., Kleb, B., Lee-Rausch, E. M., Liu, Y., Lohry, M. W., Nastac, G. C., Nielsen, E. J., Padway, E. M., Park, M. A., Rumsey, C. L., Thomas, J. L., Thompson, K. B., Walden, A. C., Wang, L., Wood, S. L., Wood, W. A., and Zhang, X., "FUN3D Manual: Version 14.2," NASA/TM 20250002308, NASA Langley Research Center, April 2025.

[23]Geuzaine, C. and Remacle, J.-F., "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities," *International journal for numerical methods in engineering*, Vol. 79, No. 11, 2009, pp. 1309–1331.

[24]Balakumar, P., Iyer, P., Nielsen, E., Anderson, W. K., Wang, L., and Diskin, B., "Assessment of UMUSCL Scheme for DNS of Turbulent Flows," *AIAA Scitech Meeting, Orlando, FL*, January 2024, AIAA Paper 2024–2199.

[25]Toosi, S. and Larsson, J., "Anisotropic grid-adaptation in large eddy simulations," *Computers & Fluids*, Vol. 156, 2017, pp. 146–161.

[26]Jovic, S. and Driver, D. M., "Reynolds number effects on the skin friction in separated flows behind a backward-facing step," *Experiments in Fluids*, Vol. 18, No. 6, 1995, pp. 464–467.