A Multi-Solver Overset Mesh Approach for 3D Mixed Element Variable Order Discretizations

Michael J. Brazell[a], Andrew Kirby[b], Jay Sitaraman[c], and Dimitri J. Mavriplis[d]

Department of Mechanical Engineering, University of Wyoming

The goal of this work is the development and demonstration of an overset mesh approach which enables arbitrary combinations of traditional second-order accurate finite-volume discretizations and variable high-order accurate discontinuous Galerkin discretizations on structured, unstructured and Cartesian meshes. The combinations of different solvers is enabled through the development of a flexible driver routine coupled with a topology independent overset grid assembly module. The approach is designed to support dynamic overset mesh problems, as well as applications with dynamic adaptive mesh refinement. The developed approach is demonstrated for sample problems running in parallel using combinations of unstructured finite-volume, unstructured discontinuous Galerkin, and Cartesian discontinuous Galerkin solvers including dynamically adaptive meshes.

I. Introduction

Overset mesh approaches have been used for many years in computational fluid dynamics due to the flexibility they afford for handling complex geometries and simulations involving bodies with relative motion such as store separation, rotorcraft, and wind turbine configurations [1–4]. Most overset mesh applications involve the use of similar mesh components, i.e. either structured meshes overlapped with structured meshes [5] or collections of overlapping unstructured meshes [6]. More recently, frameworks that support combinations of different types of overlapping meshes have been developed. One example is the HELIOS rotorcraft software [4], which supports arbitrary combinations of unstructured, structured, and cartesian meshes, along with the different discretizations used on these meshes. HELIOS applications have shown how the use of adaptively refined Cartesian meshes in off-body regions can be enabling for capturing wakes and vortices with high levels of accuracy [4, 7, 8]. High-order accurate discretizations have generated much interest over the last decade for computational aerodynamics due to the potential these methods hold for achieving higher accuracy at reduced cost and improved scalability on emerging parallel computer architectures. Naturally, the use of high-order methods in the context of overset mesh applications has been pursued by various authors who have shown that design accuracy can be maintained in an overset mesh framework for high-order discontinuous Galerkin (DG) discretizations [9–12].

However, high-order DG discretizations are best suited for flow regimes with smooth variations and limited singularities, and these methods remain more costly and less robust than traditional second-order accurate finite-volume methods for many practical aerodynamic problems. Therefore, the optimal choice of solvers for overset mesh applications may be problem dependent and in many cases combinations of low-order and high-order discretizations may be desirable.

The objective of this work is the development and demonstration of a modular flexible paradigm for combining various types of low and high-order discretizations within a common overset mesh framework. The approach currently supports combinations of finite-volume unstructured, high-order DG unstructured, and high-order DG Cartesian discretizations within a unified overset mesh framework. The framework

[a]AIAA member, Post Doctoral Research Associate, mbrazell@uwyo.edu
[b]AIAA member, Ph.D. Candidate, brezaahr@uwyo.edu
[c]AIAA member, Adjunct Professor, jsitaram@uwyo.edu
[d]AIAA Associate Fellow, Professor, mavripl@uwyo.edu

allows for arbitrary instances of each solver to run on the same or separate processor groupings in parallel, and supports dynamic mesh motion as well as dynamic adaptive mesh h- and p-refinement (AMR). The framework is also designed to enable tight coupling between different component solvers by supporting fully implicit solution methods across disparate discretizations and mesh types.

Our approach is modeled on the multi-solver paradigm developed in reference [4] as shown in Figure 1 where a near-body unstructured mesh solver is overset with an off-body Cartesian solver which supports dynamic AMR. The principal components of our approach include a driver routine which orchestrates the various solvers, the Topology Independent Overset Grid Assembler (TIOGA), and the various low- and high-order solvers which operate on the constituent meshes.

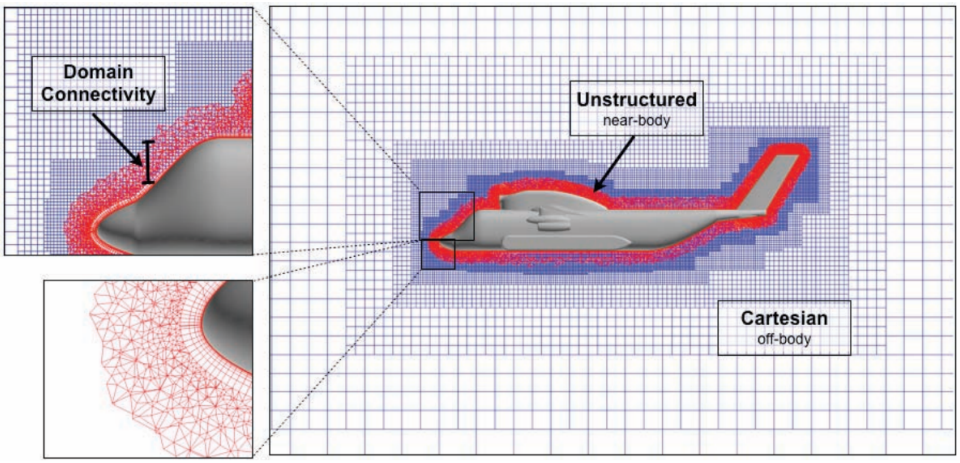


Figure 1: Example of a HELIOS dual-mesh, dual-solver overset approach [4].

In the following sections, the main driver routine is described along with details of the interface, followed by descriptions of the unstructured DG solver DG3D, the unstructured finite-volume solver NSU3D, the structured dynamic AMR DG flow solver SAMCartDG, and the TIOGA overset mesh assembler. Lastly, the feasibility of the overset framework is demonstrated with results showing verification of order of accuracy and large scale simulations of overset and dynamic AMR capabilities.

II. Driver

In a multi-solver framework a main driving routine is needed to coordinate and run each solver. The driver in this work is similar to the multi-solver framework developed by by Wissink et al. in Ref [13] with some notable differences. In this previous work, a Python interface is used to dynamically load different solvers, where each individual flow solver is typically run in sequence across all processors. This has the advantage of automatically load balancing but has some disadvantages. Each flow solver has a different amount of computational work and scales differently and in many instances concurrent execution of the different solvers on the optimal number of processors would result in more effective resource utilization. Also, high-order implicit solvers have a very large memory footprint and running each solver on the same group of processors may have significant implications for memory requirements.

The driver in this work is written in C++, handles multiple coding languages (C, C++, and Fortran) and parallel communication through MPI. It has the flexibility of running flow-solvers sequentially on the same set of processors or concurrently on different processor groups. The solvers used in this work are self contained and only communicate with each other through the TIOGA overset mesh assembler. TIOGA performs both implicit hole cutting on all meshes and executes the fringe cell interpolation between overlapping meshes. All flow solvers and other codes that are controlled by the driver will be referred to as modules. These modules can include near-body and off-body flow solvers, overset domain connectivity assemblers, as well as other disciplinary solvers such as structural solvers, mesh deformation solvers, and even atmospheric boundary layer solvers to provide turbulent inflow conditions for wind turbine simulations [7].

Each module is compiled individually using the position independent code flag (-fpic or -fPIC) to make a

shared object. These shared objects can later be used when dynamically loading libraries. The driver contains interfaces to each module and decides during runtime which dynamic library to load. The interfaces on the module side use C-binding. For C++ this is enabled by using extern "C" in front of interface functions, which also avoids name mangling. Name mangling is a feature that compilers use to create unique names in the shared object for linking to later on. To use these functions inside the shared object one must manually search for their assigned names, which can be compiler dependent. C-binding forces the compiler to use the specified name and makes it easier to find these functions in the shared object. In Fortran C-binding is done by using the ISO_C_BINDING module which gives access to C pointers (C_PTR) and C function pointers (C_FUNPTR) and also allows the name of the routine to be specified in the shared object. This is helpful because interface functions that are located inside Fortran modules have names that are compiler dependent and are combinations of the module name and the function name. Below is an example of using C-Binding in a F90 routine.

```
module driver_interface_module

    contains

    subroutine set_pointers(soln_ptr, &
                            iblank_ptr, &
                            callback_function_ptr)  bind(C,name="driver_interface_set_pointers")

        use solution_module, only : soln,iblank
        use iso_c_binding

        implicit none

        type(c_ptr),intent(out) :: soln_ptr
        type(c_ptr),intent(out) :: iblank_ptr
        type(c_funptr),intent(out) :: callback_function_ptr

        soln_ptr = c_loc(soln(1))
        iblank_ptr = c_loc(iblank(1))
        callback_function_ptr = c_funloc(count_receptor_nodes)

    end subroutine

end module
```

The driver can handle an arbitrary number of groups and groupings of processors which are specified all through input files. Figure 2 shows an outline of a simple driver using two near-body solvers and one off-body solver connected using TIOGA. In this case at every time step the near-body mesh undergoes a prescribed motion and the off-body performs a mesh adaption step.

III. Code Modules

This section describes the code modules incorporated into the driver. The first two are unstructured near body solvers DG3D and NSU3D, the third is a structured off body solver SAMCartDG, and the fourth is the overset mesh connectivity assembler TIOGA.

A. Unstructured near-body DG solver: DG3D

The first near-body solver is a parallel 3D Discontinuous Galerkin (DG) finite element method. The solver can handle hybrid mixed element meshes (tetrahedra, pyramids, prisms, and hexahedra), curved elements, and incorporates both p-enrichment and h-refinement capabilities using non-conforming elements (hanging

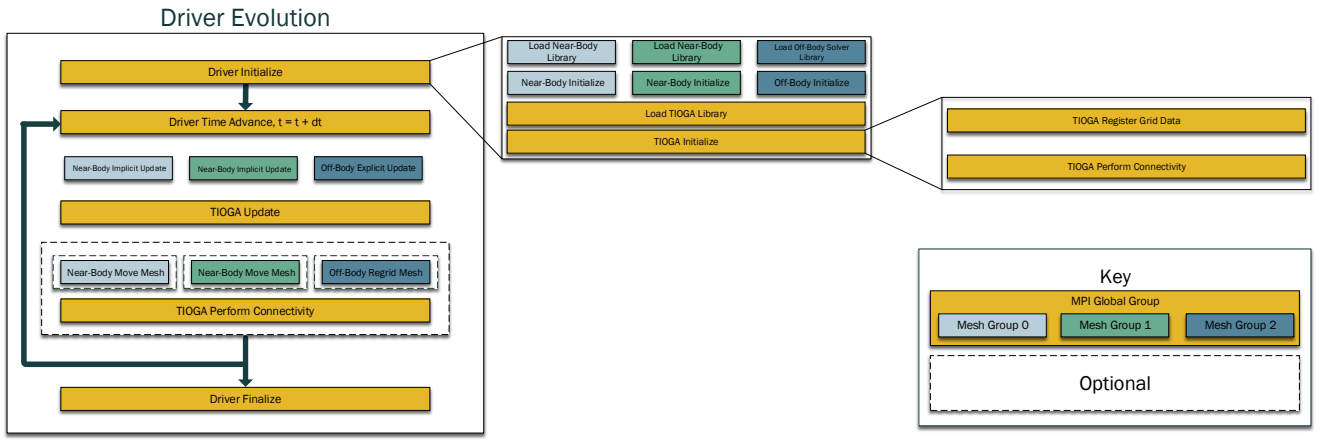


Figure 2: Outline of an unsteady driver example with mesh movement and adaption.

nodes). It is used to discretize the compressible Navier-Stokes equations as well a PDE-based artificial viscosity equation [14–16] and the Spalart-Allmaras turbulence model (negative-SA variant) [17]. The advective fluxes are calculated using a Riemann solver. Implemented Riemann solvers include: Lax-Friedrichs [18], Roe [19], and artificially upstream flux vector splitting scheme (AUFS) [20]. The diffusive fluxes are handled using a symmetric interior penalty (SIP) method [21, 22]. The time derivative can be approximated using the explicit scheme RK4 or an implicit BDF2 scheme. The implicit solver uses a Newton-Raphson method to solve the non-linear set of equations. These equations are linearized to obtain the full Jacobian. The linear system is solved using a flexible-GMRES [23] (fGMRES) method. To further improve convergence of fGMRES a right preconditioner can be applied to the system of equations. Preconditioners that have been implemented include Jacobi relaxation, Gauss-Seidel relaxation, line implicit Jacobi, and ILU(0). This solver has been used to successfully to solve hypersonic flows [24], turbulent flow over wings [25] and aircraft, Direct Numerical Simulations (DNS) of Taylor Green Vortex, and overset simulations [10].

B. Unstructured near-body finite volume solver: NSU3D

A second near-body solver, NSU3D, is an unstructured mesh multigrid Unsteady Reynolds-averaged Navier-Stokes (URANS) solver developed for high-Reynolds number external aerodynamic applications. NSU3D employs a second-order accurate vertex-based discretization, where the unknown fluid and turbulence variables are stored at the vertices of the mesh, and fluxes are computed on faces delimiting dual control volumes, with each dual face being associated with a mesh edge. This discretization operates on hybrid mixed-element meshes, generally employing prismatic elements in highly stretched boundary layer regions, and tetrahedral elements in isotropic regions of the mesh. A single edge-based data structure is used to compute flux balances across all types of elements. The single-equation Spalart-Allmaras turbulence model [26] as well as a standard $k-\omega$ two-equation turbulence model [27] are available within the NSU3D solver.

The NSU3D solution scheme was originally developed for optimizing convergence of steady-state problems. The basic approach relies on an explicit multistage scheme which is preconditioned by a local block-Jacobi preconditioner in regions of isotropic grid cells. In boundary layer regions, where the grid is highly stretched, a line preconditioner is employed to relieve the stiffness associated with the mesh anisotropy [28]. An agglomeration multigrid algorithm is used to further enhance convergence to steady-state [29, 30]. The Jacobi and line preconditioners are used to drive the various levels of the multigrid sequence, resulting in a rapidly converging solution technique.

For time-dependent problems, first-, second-, and third-order implicit backwards difference time discretizations are implemented, and the line-implicit multigrid scheme is used to solve the non-linear problem arising at each implicit time step. NSU3D has been extensively validated in stand-alone mode, both for steady-state fixed-wing cases, as a regular participant in the AIAA Drag Prediction workshop series [31], as well as for unsteady aerodynamic and aeroelastic problems, [32] and has been benchmarked on large parallel

computer systems [33].

C. Cartesian Off-body Solver: SAMCartDG

The off-body solver is a combination of a block-structured Cartesian high-order discontinuous Galerkin solver known as CartDG [34], and an open source adaptive mesh refinement (AMR) framework called SAMRAI developed at the Lawrence Livermore National Lab [35,36]. The off-body solver is referred to as SAMCartDG [34]. SAMRAI uses a block-structured grid hierarchy of nested refinement levels. Each level is composed of a union of logically-rectangular grid regions. On each level the grid spacing is fixed and the ratio of refinement between two consecutive levels is generally two, although the environment allows for other refinement ratios pending the construction of appropriate refinement and coarsening operators. SAMRAI performs all AMR grid generation, load balancing, and parallel communications between blocks. If regridding is requested, cells are tagged for refinement using a feature detection criteria implemented by the user. The current tagging algorithm is based on vorticity magnitude.

1. CartDG: Cartesian High-Order Block Solver

The block solver is a discontinuous Galerkin Finite Element Method based compressible Navier-Stokes solver. It is developed for an arbitrary order of accuracy. The stand alone solver has been demonstrated for various inviscid and viscous problems, most notably the Ringleb flow [37] and the Taylor-Green Vortex [38], respectively.

CartDG's parallel scalability up to 32,768 processors has been previously demonstrated [34] on the NCAR Wyoming supercomputer named Yellowstone [39]. Further parallel scalability tests have been conducted on Argonne Leadership Computing Facility's (ALCF) Mira Supercomputer which has 786,432 processors capable of 10 petaflops peak performance. Mira is an IBM Blue Gene/Q system composed of 1600 MHz PowerPC A2 cores with a 5D torus interconnect.

A strong scalability test was conducted using a range of processors from 8,192 to 524,288 processors with one MPI rank per core and two MPI ranks per core. The strong scalability results are obtained by performing a simulation of the fully periodic Taylor-Green vortex problem using a 512 x 512 x 512 mesh at $p = 4$, fifth order accuracy. The problem contains approximately 16.8 billion degrees-of-freedom running for 5 time steps using RK4. Tables 1 and 2 present the execution times for one MPI rank per core and two MPI ranks per core, respectively. Figure 3 demonstrates the time to solution and the strong scaling percentage up to 524,288 processors using a one and two MPI ranks per processor. The strong scaling results assume the time to solution for 8,192 processors is ideal. We note that executing two MPI ranks per core is approximately 64% faster in execution time in comparison to one MPI rank per processor. This trend holds to over one million MPI ranks.

Table 1: Mira Scalability: One MPI Rank per Core.

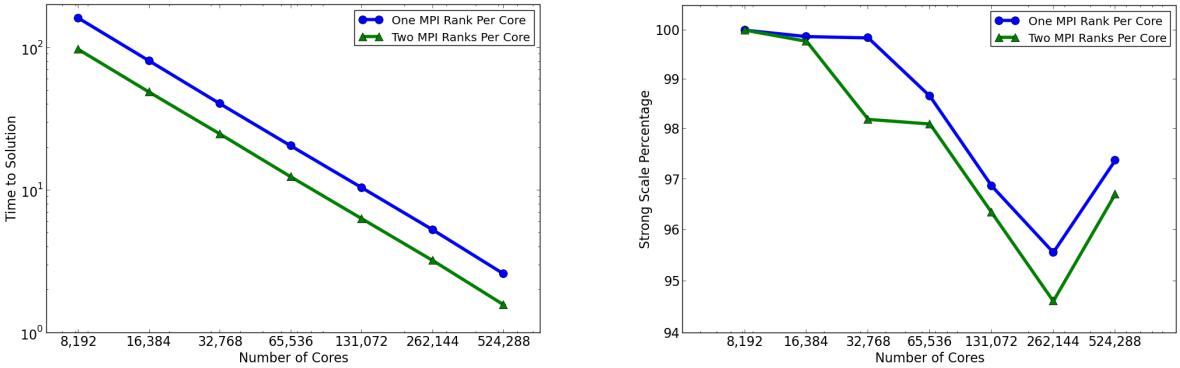
Cores	Time (sec)	Strong Scalability %	Speedup
8,192	161.32199	100.00%	8192
16,384	80.76674	99.869%	16362.54
32,768	40.39420	99.842%	32716.23
65,536	20.43908	98.660%	64657.82
131,072	10.40850	96.869%	126968.14
262,144	5.27547	95.561%	250507.43
524,288	2.58876	97.369%	510493.98

D. Overset Mesh Assembly: TIOGA

The overset grid connectivities are handled through the TIOGA interface (Topology Independent Overset Grid Assembler) [10]. The TIOGA overset grid assembler relies on an efficient parallel implementation of Alternating Digital Trees (ADT) [40] for point-cell inclusion tests. Multiple grids are loaded in parallel and TIOGA computes the IBLANKing information required by the flow solver, as well as the cell donor-receptor

Table 2: Mira Scalability: Two MPI Ranks per Core.

Cores	MPI Ranks	Time (sec)	Strong Scalability %	2:1 Ranks Time Ratio
8,192	16,384	97.297702	100.00%	1.658
16,384	32,768	48.75938	99.773%	1.656
32,768	65,536	24.77189	98.194%	1.631
65,536	131,072	12.39752	98.102%	1.649
131,072	262,144	6.311845	96.344%	1.649
262,144	524,288	3.21362	94.614%	1.642
524,288	1,048,576	1.57219	96.698%	1.647



(a) Time to Solution

(b) Strong Scaling Percentage

Figure 3: CartDG strong scalability on ALCF's Mira supercomputer up to 524,288 processors on a problem containing nearly 16.8 billion degrees-of-freedom.

information. We use the notation IBLANK which is an array of integers used to identify cell type in the overset framework. In this notation a fringe cell has IBLANK value of: IBLANK $= -1$, a hole cell: IBLANK $= 0$, and a field cell: IBLANK $= 1$. A hole cell occurs when an outer mesh overlaps an inner mesh with an enclosed surface (such as a sphere). This hole cell does not contain a donor cell and is usually eliminated from the discretization. This is allowed because a collection of hole cells needs to be completely surrounded by fringe cells. This in turn means that the field cells only interact with fringe cells. If a field cell neighbors a hole cell then the overset connectivity fails. The receptor cells receive solution data from the donor cells through interpolation as shown in Figure 4. In order to interface TIOGA with a high-order method several callback functions need to be supplied. This allows high-order interpolation which is required to maintain overall high-order solution accuracy.

There are four callback functions that are needed to interface TIOGA with a high-order method. The first callback function is a function which generates a list of receptor points. Typically these are the nodes or cell centers in a mesh, but for a high-order discretization additional points are needed within each cell. The second function is a high-order donor inclusion test which is necessary for curved elements. Once donor cells are located the third function returns the weights for the high-order interpolation on the donor cells. The last callback function converts the interpolated solution at the receptor points back into solution coefficients or modal coefficients on the receptor cell.

There are two approaches for transferring solution data: the first uses a mass matrix approach and the second a Vandermode matrix approach. One of these two methods is necessary because it is the transfer of solution coefficients rather than the actual solution values that are needed in a modal finite-element method. Both the mass matrix and the Vandermode matrix methods require solution values from the donor cells at specific points on the receptor cell. Figure 5 shows the locations of the receptor nodes for each method. For

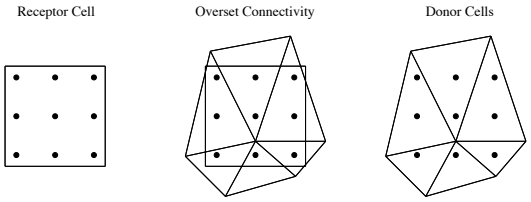


Figure 4: Receptor and donor cell overset connectivity.

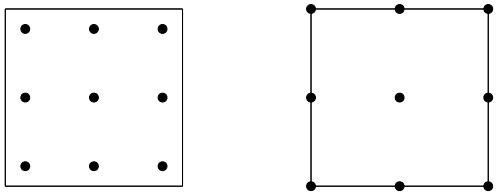


Figure 5: Locations of receptor nodes on a quadrilateral element for mass matrix method (left) and Vandermode matrix method (right).

the mass matrix method the interpolation points correspond to the quadrature points used in the volume integral evaluation of the residual, while for the Vandermode matrix method the interpolation points consist of equidistant points. The interpolation points become the receptor points in the first callback function and then the receptor nodes are used by TIOGA to determine the donor cells.

To find the donor cells a callback function for a high-order curved cell inclusion test is used. TIOGA uses its ADT to search for donor cells and the callback function is used to determine if the receptor point is located inside a donor cell. For straight sided elements TIOGA can handle this process using straight forward geometric inclusion tests. However for a curved element, a specialized callback function is needed. To test if a point is inside an element, the physical coordinates (x, y, z) are converted to natural coordinates (r, s, t) in the mapped space of the standard isoparametric element, as shown in Figure 6. The natural coordinates are found using the following equations:

$$\sum_{m=1}^{n_{map}} \psi_m(r,s,t) b_{1m} = x, \quad \sum_{m=1}^{n_{map}} \psi_m(r,s,t) b_{2m} = y, \quad \sum_{m=1}^{n_{map}} \psi_m(r,s,t) b_{3m} = z,$$

where b are the physical coordinate mapping coefficients, ψ are the mapping basis functions, and n_{map} are the number of mapping coefficients for the element. This gives three non-linear equations with three unknowns. To solve this problem a Newton-Rhapson method is used which can be written as:

$$\begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial s} & \frac{\partial z}{\partial t} \end{bmatrix} \begin{bmatrix} \delta r \\ \delta s \\ \delta t \end{bmatrix} = - \begin{bmatrix} \sum_{m=1}^{n_{map}} \psi_m(r,s,t) b_{1m} - x \\ \sum_{m=1}^{n_{map}} \psi_m(r,s,t) b_{2m} - y \\ \sum_{m=1}^{n_{map}} \psi_m(r,s,t) b_{3m} - z \end{bmatrix}$$

Once the natural coordinates are found it is trivial to test if the point lies inside the element. For example, a tetrahedra has four faces that are each associated with a plane that can be tested for inclusion. If any of the following inequalities in Table 3 are true then the point does not lie inside the element. In this table, the parameter ϵ is a small number, if this number approaches zero then strict inclusion is required. However, if

this number is larger, the inclusion test will be less restrictive and extrapolation (instead of interpolation) is allowed.

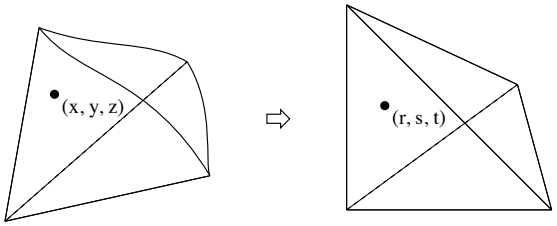


Figure 6: Curved tetrahedron in physical coordinates (left) and natural coordinates (right)

tetrahedron	pyramid	prism	hexahedron
$r+1<-\epsilon$	$r+1<-\epsilon$	$r+1<-\epsilon$	$\lvert r\rvert-1>\epsilon$
$s+1<-\epsilon$	$s+1<-\epsilon$	$\lvert s\rvert-1>\epsilon$	$\lvert s\rvert-1>\epsilon$
$t+1<-\epsilon$	$t+1<-\epsilon$	$t+1<-\epsilon$	$\lvert t\rvert-1>\epsilon$
$r+s+t>\epsilon$	$r+t>\epsilon$	$r+t<-\epsilon$	
	$s+t>\epsilon$		

Table 3: Inequalities for each element type for inclusion.

Once a donor cell is found the third callback function creates the interpolation weights. Evaluating the solution basis functions at the natural coordinates found from the inclusion test creates a high-order interpolation for the donor cell. The solution q is passed back to the receptor cell by:

$$q\left(r,s,t\right)=\sum_{m=1}^{n_{mode}}\phi_m^d\left(r,s,t\right)a_m$$

where $\phi_m^d\left(r,s,t\right)$ are the donor cell basis functions (where superscript d represents donor) evaluated at the receptor node locations, n_{mode} are the number of solution basis functions, and a_m are the solution coefficients on the donor cell. TIOGA transfers, in parallel, all of the interpolated solutions from the donor cells to the receptor cells. The fourth callback function converts these solution values back into coefficients on the receptor cells. As discussed previously two approaches have been implemented; the first is the mass matrix method. This approach starts off with a Galerkin projection:

$$\sum_{n=1}^{n_{mode}}\phi_n^r(\xi_{1k},\xi_{2k},\xi_{3k})a_n=q(\xi_{1k},\xi_{2k},\xi_{3k})$$

where ϕ_n^r are the solution receptor basis functions (where superscript r represents receptor), n_{mode} is the number of solution receptor basis functions, ξ_{ik} are the quadrature points (where the indices $i=1,2,3$ and k runs from one to the number of quadrature points n_{qp}), q are the solution values, and a_n are the receptor solution coefficients to be solved for. To solve for the coefficients both sides are multiplied by the receptor cells basis function and integrated over the cell:

$$\sum_{n=1}^{n_{mode}}\int_\Omega\phi_m^r\phi_n^r a_n\,\mathrm{d}\Omega=\int_\Omega\phi_m q\mathrm{d}\Omega.$$

This creates a mass matrix defined as:

$$M_{mn}=\int_\Omega\phi_m^r\phi_n^r\mathrm{d}\Omega$$

and the right hand side becomes:

$$f_m = \int_\Omega \phi_m q \mathrm{d}\Omega = \sum_{k=1}^{n_{qp}} \phi_m(\xi_{1k}, \xi_{2k}, \xi_{3k}) q(\xi_{1k}, \xi_{2k}, \xi_{3k}) w_k$$

where w_k are the quadrature weights. To solve for the coefficients the right hand side is integrated and the mass matrix is inverted to give:

$$a = \mathbf{M}^{-1} f.$$

The mass matrix is LU factorized ahead of time and only a forward/backward solve is needed to solve for a.

The second approach is the Vandermode method. Again, this starts with a projection operator:

$$\sum_{n=1}^{n_{mode}} \phi_n(\zeta_{1m}, \zeta_{2m}, \zeta_{3m}) a_n = q(\zeta_{1m}, \zeta_{2m}, \zeta_{3m})$$

except now the Vandermode matrix:

$$V_{mn} = \phi_n(\zeta_{1m}, \zeta_{2m}, \zeta_{3m})$$

is constructed by evaluating the basis at the points ζ_{im}, where the number of points are equal to the number of basis functions, making the Vandermode matrix square. The solution coefficients are solved for by inverting the Vandermode matrix:

$$a = \mathbf{V}^{-1} q$$

for the receptor solution coefficients. For efficiency the Vandermode matrix is constructed for every element type and polynomial degree and factorized ahead of time so that only forward/backward solves are needed.

There are two downsides to the mass matrix method. The first is that even if the mass matrix is diagonal the right hand side still requires integration with respect to the basis. The second is that, for tetrahedra, pyramids, and prisms, the number of quadrature points is greater than the number of basis modes. Also, for curved elements more quadrature points are needed making the mass matrix approach even more costly. Due to this the mass matrix method requires more computational work than the Vandermode matrix method. However, a benefit of the mass matrix method is that it gives a more optimal approximation to the interpolated values compared to the Vandermode approach at equidistant points. The Vandermode matrix at equidistant points can suffer from interpolation errors at high polynomial degrees.

For SAMCartDG a simplification occurs due to the nodal Lagrange polynomial basis formed using the Gauss Legendre points. This set of points is used for either the Vandermode or mass matrix approaches and both give the same end result. In the case of the Vandermode matrix approach, each basis function equals one at its corresponding quadrature point and zero at every other quadrature points. This in turn creates an identity matrix for the Vandermode matrix. For the mass matrix approach the number of quadrature points equals the number of modes and the mass matrix is diagonal for this basis. The nodal basis simplifies the right hand integration so that it is performed at only one quadrature point. This means the mass matrix can be inverted and absorbed into the right hand side integration giving an identical result to the Vandermode approach.

IV. Results

Three test cases are chosen to demonstrate the ability to simulate unsteady separated flows in an overset framework. The first case is a low Reynolds number, DNS of flow over a sphere. This case tests the ability to combine two high-order DG codes by using the unstructured DG solver on the near-body and the AMR DG solver on the off-body. The second case is a simulation of a higher Reynolds number turbulent flow over a rectangular lifting wing. This case tests the ability to combine different discretizations, one being a node based finite-volume RANS solver and the other being a cell-based AMR DG solver, and to capture and preserve wing tip vortices. The last case consists of the simulation of turbulent flow over the 5 MW NREL wind turbine, again using the finite-volume RANS solver and the AMR DG solver. This case demonstrates the capability of simulating flows with bodies in relative motion as well as capturing vortical structures and wakes. All three of these cases showcase the ability to simulate problems in an overset framework with the dynamic mesh adaptation in the off-body.

A. DNS of flow over a sphere

The first case is a simulation of an unsteady, low Reynolds number, DNS flow over a sphere. Flow over a sphere is a good validation case because it has been studied extensively in both experiments and numerical simulations. For this particular simulation the free-stream Mach number is $M = 0.3$ and the Reynolds number is $Re_D = 1000$, based on sphere diameter. This case combines the near-body unstructured DG solver with the off-body AMR DG solver using an overset mesh. The near-body mesh is a strand grid which consists of prismatic cells. This grid is created by extruding (radially outward) an unstructured triangular mesh on the surface of the sphere. The near-body DG solver is third order accurate ($p = 2$) in space and is solved on 64 cores. The off-body DG solver is fourth order accurate ($p = 3$) in space and solved on 256 cores. Four levels of grid refinement are used in this simulation. Figure 7 shows the wake created by the sphere as iso-contours of vorticity magnitude. Also, shown is the adaptive off-body grid which closely tracks the vortices in the wake as they travel downstream.

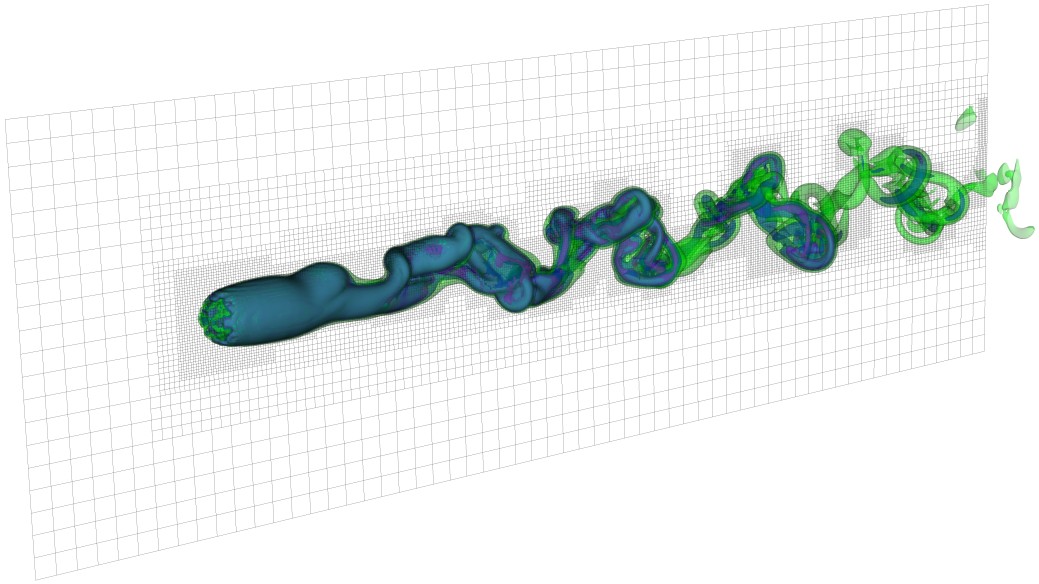


Figure 7: Iso-contours of vorticity magnitude for flow over a sphere with $Re_D = 1000$.

Validation of this case is performed by comparing the drag coefficient on the sphere to both experimental data and other simulations. Since the flow is unsteady the drag coefficient is time averaged. The time history of the drag coefficient is shown in Figure 8a along with a running average. Also shown in Figure 8b is a close up view of the final running average. The running average is calculated using a moving average with a bin size of 2.5×10^5 time steps. This averaging procedure ensures that the initial transients do not adversely affect the time-averaged drag coefficient of the fully developed flow. The final time averaged drag coefficient

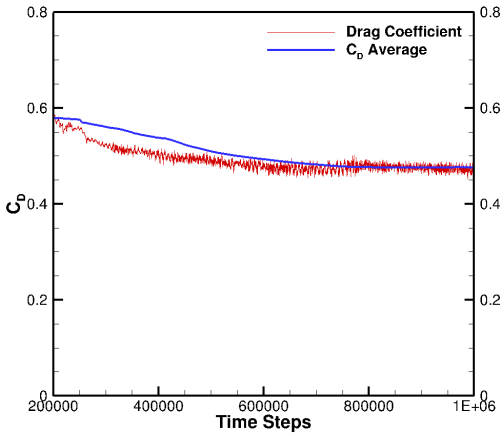
Table 4: Drag coefficient compared with data from literature.

Re_D	Present	Ref. [41]	Ref. [42]	Eqn.1
1000	0.475	0.4818	0.476	0.4841

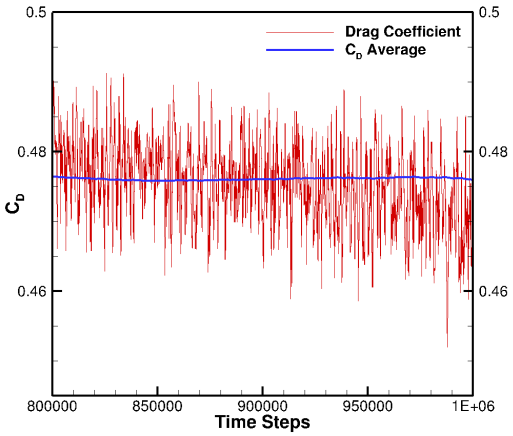
is shown in Table 4 along with experimental data, simulation data, and data correlation. The correlation for the drag coefficient in uniform flow around a sphere is given by the following equation:

$$C_D = \frac{24}{Re_D} + \frac{2.6\left(\frac{5.0}{Re_D}\right)}{1+\left(\frac{5.0}{Re_D}\right)^{1.52}} + \frac{0.411\left(\frac{Re_D}{263{,}000}\right)^{-7.94}}{1+\left(\frac{Re_D}{263{,}000}\right)^{-8.0}} + \left(\frac{Re_D^{0.80}}{461{,}000}\right) \tag{1}$$

where Re_D is the Reynolds number based on diameter of the sphere [43]. The presented drag coefficient is very similar to the referenced numerical simulation [42] and reasonably close to the experimental data [41].



(a) Time history of drag coefficient and running average.



(b) Close-up view of running average.

Figure 8: Drag history and running average of flow over a sphere, $Re_D = 1000$ and $Mach = 0.3$.

This demonstrates that the DG solvers combined in an overset framework are capable of accurately simulating unsteady, low Reynolds number flow around simple geometries. Also, high-order methods combined with AMR result in a very large reduction in the number of degrees of freedom necessary to solve problems of this type.

B. NACA 0015 wing

The second case demonstrates the capability of simulating unsteady, turbulent flow over a rectangular square-tipped lifting wing. The wing is based on a constant cross-section NACA0015 wing, and has a finite span of 6.6 chords. For this case the Mach number is 0.1235 and the Reynolds number is 1.5 million (based on chord length). NSU3D is used on the near-body and the AMR DG solver is used on the off-body. This case is used to demonstrate the ability to combine a node-based finite-volume discretization with a cell-based, high-order, DG discretization in an overset framework, and to accurately capture tip vortices. NSU3D solves the unsteady RANS equations closed by the Spalart-Allmaras with Rotation Correction (SA-RC) turbulence model. The time discretization for NSU3D is the implicit, second-order accurate BDF2 scheme. The off-body AMR DG solver uses 9 levels of refinement and a second order ($p = 1$) discretization in space and second order explicit discretization in time (RK2). The NACA 0015 wing has been studied experimentally by McAlister and Takhashi [44]. Computational studies have been performed by Wissink [45], Sitaraman and Baeder [46], and by Hariharan and Sankar [47]. Figure 9 shows the time evolution of lift and drag on the wing for an inflow at 10 degrees incidence. At these conditions, the experimentally reported lift and drag coefficients are 0.88 and 0.05, respectively [44], which are in good agreement with the final average time history values in Figure 9. The finite-span lifting wing generates counter rotating tip vortices which are visualized as iso-surfaces of vorticity magnitude as shown in Figure 10. Using AMR, these wing tip vortices can be resolved and accurately tracked downstream.

To determine how accurately the vortices are being captured, the vertical velocity can be compared to experimental values [44] downstream. These comparisons are made for the case where the wing is at 12 degrees incidence to the incoming flow. At these conditions, the simulation resulted in flow separation on the wing after a long time integration period. Therefore, the comparison is made at a point in time prior to the occurrence of flow separation, when the lift coefficient reached a value of 1.15, which is close the the experimentally reported value for this incidence. Figure 11 shows the vertical velocity of the simulation compared with the experiment at 2, 4, and 6 chords downstream from the wing. These comparisons are made in the spanwise direction (y-dir) at $z/C = -0.1$, where C is the chord length. There is about 40% error in the downstream vortex velocity strength. The weaker vortex could be attributed to having to pass through the

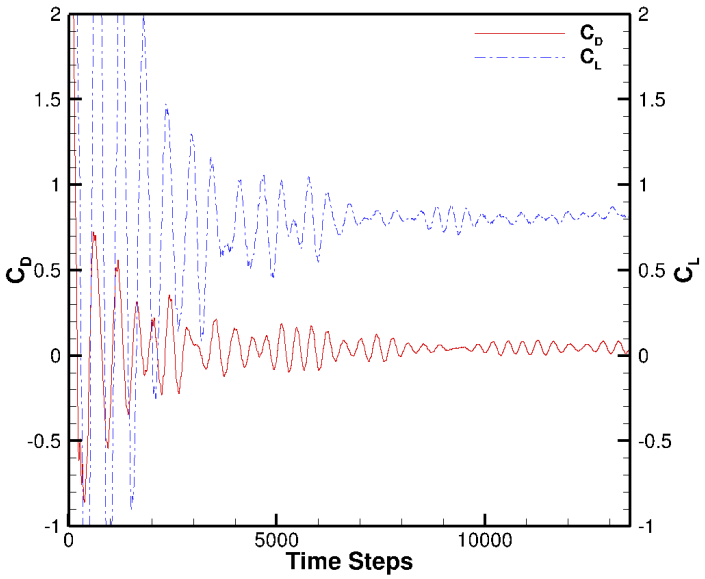


Figure 9: Lift and drag history of NACA 0015 wing computation at 10 degrees incidence for a Mach number of 0.1235 and Reynolds number 1.5 million.

near-body grid and dissipating before it enters the off-body grid. This level of agreement is similar to that reported in reference [45] using a finite-difference off-body AMR strategy with the same near-body mesh.

C. 5MW NREL Wind Turbine

The final case is a large scale simulation of the 5MW NREL wind turbine [48]. The inflow velocity is 11.4 m/sec, which corresponds to a free-stream Mach number is 0.033, the Reynolds number is 1.77×10^7 based on a mean chord length of 3.493 meters, and the blades spin at a prescribed rate of 12 rpm. This case is a demonstration of combining two discretizations (finite-volume node-based on the near body and AMR DG cell based on the off-body) in an overset framework with bodies moving in relative motion. The near-body solver is NSU3D and it handles the tower and the three spinning blades. The unsteady RANS equations are solved and the time discretization is a GCL compliant BDF2 scheme. Each of the 4 grids (3 blades and tower) are given 16 cores for a total of 64 near-body cores. The off-body solver is SAMCartDG running at second order accuracy in space and time. It is solved on 2048 cores and allows for 7 levels of mesh refinement. Figure 12 shows iso-contours of vorticity magnitude and a slice of the AMR grid. After one full rotation of the blades there are about 10 million elements in the off-body solver (80 million DOF's per solution variable). The tip vortices are well captured and the AMR solver tracks these downstream.

V. Conclusions

We have developed and demonstrated a general framework for combining different discretizations on different mesh types for dynamic and adaptive overset mesh problems. The coupling of the different solvers is managed by a driver routine and the overset mesh connectivities through TIOGA. The capability of this framework was demonstrated first on low Reynolds number flow over a sphere. The drag coefficient for this problem matches closely to experiments. The second case involves the turbulent flow around a NACA 0015 wing. This simulation demonstrates the ability to capture wing tip vortices very far downstream using the AMR DG solver. The final case is a large scale demonstration of a full 5MW NREL wind turbine. This demonstrates the capability of handling mixed discretizations and multiple moving grids. The tip vortex is captured and advected downstream accurately due to the AMR DG solver. This capability would not be possible without using a multi-solver approach. Each solver is highly specialized and when combined allow

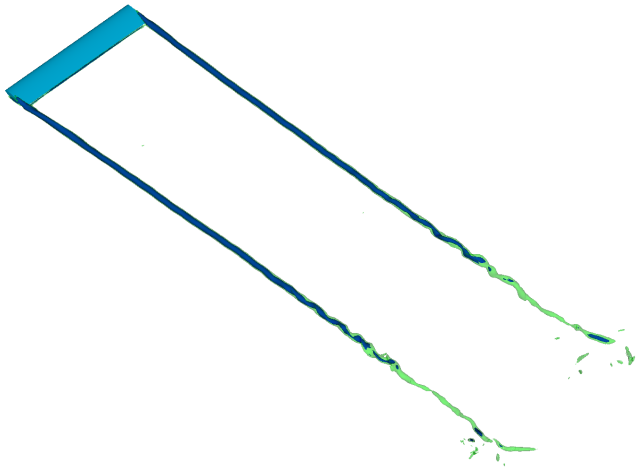
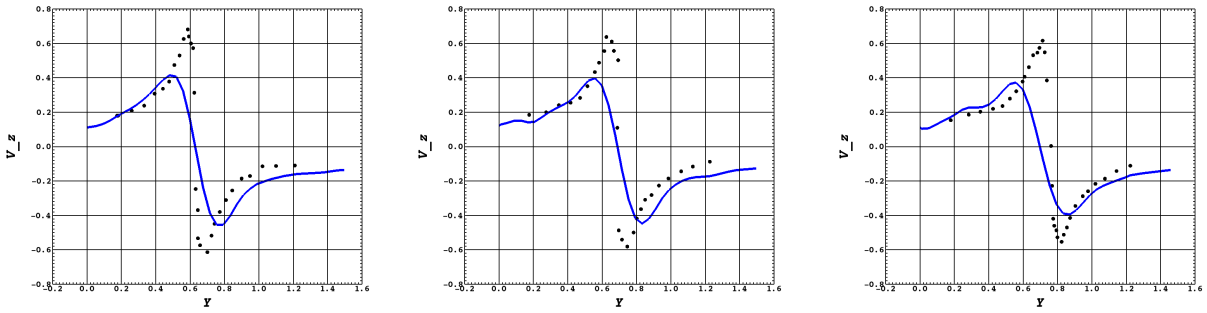


Figure 10: Iso-surfaces of vorticity magnitude for flow over NACA0015 wing at 12 degrees incidence and Mach number of 0.1235 and Reynolds number 1.5 million.



(a) 2 chords downstream. (b) 4 chords downstream. (c) 6 chords downstream.

Figure 11: Vertical velocity of simulation (solid blue line) and experimental data (black symbols) at 2, 4, and 6 chords downstream of NACA 0015 wing at $z/C = -0.1$.

for simulations that are larger, more realistic, and more accurate than previously possible. Future work is underway to demonstrate this capability using much higher orders of accuracy in the off-body region. This requires the implementation of an h-p adaptive refinement strategy in the off-body region, which is currently under development.

VI. Acknowledgments

This work was supported in part by ONR Grant N00014-14-1-0045 and by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, under Award DE-SC0012671. Computer time was provided by the NCAR-Wyoming Supercomputing Center (NWSC) and University of Wyoming Advanced Research Computing Center (ARCC).

References

[1]Murphy, K. J., Buning, P. G., Pamadi, B. N., Scallion, W. I., and Jones, K. M., "Overview of Transonic to Hypersonic Stage Separation Tool Development for Multi-Stage-To-Orbit Concepts," *AIAA Paper*, Vol. 2595, 2004, pp. 2004.

[2]Brown, D. L. and Henshaw, W. D., "Overture: An Object-Oriented Framework for Solving Partial Differential Equations on Overlapping Grids," *Object Oriented Methods for Interoperable Scientific and Engineering Computing, SIAM*, 1999, pp. 245–255.

[3]Noack, R., "SUGGAR: a general capability for moving body overset grid assembly," *AIAA paper*, Vol. 5117, 2005, pp. 2005.

[4]Wissink, A., Kamkar, S., Pulliam, T., Sitaraman, J., and Sankaran, V., "Cartesian Adaptive Mesh Refinement for Rotorcraft Wake Resolution," *AIAA, 28th Applied Aerodynamics Conference*, 2010, AIAA Paper 2010-4554, 28th AIAA Applied Aerodynamics Conference, Chicago, IL, June 2010.

[5]Buning, P. G., Gomez, R. J., and Scallion, W. I., "CFD Approaches for Simulation of Wing-Body Stage Separation," *AIAA Paper*, Vol. 4838, 2004, pp. 2004.

[6]Biedron, R. T. and Thomas, J. L., "Recent Enhancements to the FUN3D Flow Solver for Moving-Mesh Applications," *AIAA Paper*, Vol. 1360, 2009, pp. 2009.

[7]Sitaraman, J., Mavriplis, D. J., and Duque, E. P., *Wind Farm simulations using a Full Rotor Model for Wind Turbines*, American Institute of Aeronautics and Astronautics, 2015/05/29 2014.

[8]Sankaran, V., Sitaraman, J., Wissink, A., Datta, A., Jayaraman, B., Potsdam, M., Mavriplis, D., Yang, Z., O'Brien, D., Saberi, H., et al., "Application of the Helios Computational Platform to Rotorcraft Flowfields," *AIAA paper*, Vol. 1230, 2010, pp. 2010.

[9]Galbraith, M. C., Benek, J. A., Orkwis, P. D., and Turner, M. G., "A Discontinuous Galerkin Chimera scheme," *Computers & Fluids*, Vol. 98, No. 0, 2014, pp. 27 – 53.

[10]Brazell, M. J., Mavriplis, D. J., and Sitaraman, J., *An Overset Mesh Approach for 3D Mixed Element High Order Discretizations*, American Institute of Aeronautics and Astronautics, 2015/05/27 2015.

[11]Zhang, B. and Liang, C., *A Simple, Efficient, High-Order Accurate Sliding-mesh Interface Approach to FR/CPR Method on Coupled Rotating and Stationary Domains*, American Institute of Aeronautics and Astronautics, 2015/06/01 2015.

[12]Merrill, B. and Peet, Y., *High-Order Moving Overlapping Grid Methodology for Aerospace Applications*, American Institute of Aeronautics and Astronautics, 2015/06/01 2015.

[13]Wissink, A., Sitaraman, J., Sankaran, V., Mavriplis, D., and Pulliam, T., *A Multi-Code Python-Based Infrastructure for Overset CFD with Adaptive Cartesian Grids*, American Institute of Aeronautics and Astronautics, 2015/05/27 2008.

[14]Persson, P.-O. and Peraire, J., "Sub-cell shock capturing for discontinuous Galerkin methods," *Collection of Technical Papers - 44th AIAA Aerospace Sciences Meeting*, Vol. 2, 2006, pp. 1408 – 1420.

[15]Burgess, N., *An Adaptive Discontinuous Galerkin Solver for Aerodynamic Flows*, Ph.D. thesis, University of Wyoming, 2011.

[16]Barter, G. and Darmofal, D., "Shock capturing with PDE-based artificial viscosity for DGFEM: Part I. Formulation," *J. Comput. Phys. (USA)*, Vol. 229, No. 5, 2010/03/01, pp. 1810 – 27.

[17]Allmaras, S., Johnson, F., and Spalart, P., "Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model," *7th International Conference on Computational Fluid Dynamics*, 2012.

[18]Lax, P. D., "Weak solutions of nonlinear hyperbolic equations and their numerical computation," *Communications on Pure and Applied Mathematics*, Vol. 7, No. 1, 1954, pp. 159–193.

[19]Roe, P., "Approximate Riemann solvers, parameter vectors, and difference schemes," *J. Comput. Phys. (USA)*, Vol. 43, No. 2, 1981/10/, pp. 357 – 72.

[20]Sun, M. and Takayama, K., "An artificially upstream flux vector splitting scheme for the Euler equations," *J. Comput. Phys. (USA)*, Vol. 189, No. 1, 2003/07/20, pp. 305 – 29.

[21]Hartmann, R. and Houston, P., "An optimal order interior penalty discontinuous Galerkin discretization of the compressible Navier-Stokes equations," *J. Comput. Phys. (USA)*, Vol. 227, No. 22, 2008/11/20, pp. 9670 – 85.

[22]Shahbazi, K., Mavriplis, D., and Burgess, N., "Multigrid algorithms for high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations," *J. Comput. Phys. (USA)*, Vol. 228, No. 21, 2009/11/20, pp. 7917 – 40.

[23]Saad, Y., "A flexible inner-outer preconditioned GMRES algorithm," *SIAM J. Sci. Comput.*, Vol. 14, No. 2, March 1993, pp. 461–469.

[24]Brazell, M. J. and Mavriplis, D. J., *3D Mixed Element Discontinuous Galerkin with Shock Capturing*, AIAA Paper 2013-2855, 21st AIAA CFD Conference, San Diego, CA, June 2013.

[25]Brazell, M. J. and Mavriplis, D. J., *High-Order Discontinuous Galerkin Mesh Resolved Turbulent Flow Simulations of a NACA 0012 Airfoil (Invited)*, AIAA Paper 2015-1529, 53rd AIAA Aerospace Sciences Meeting, Kissimmee, FL, January 2015.

[26]Spalart, P. R. and Allmaras, S. R., "A one-equation turbulence model for aerodynamic flows," *La Recherche A erospatiale*, Vol. Vol. 1, 1994, pp. 5–21., No. 5-21, 1994.

[27]Wilcox, D. C., "Reassessment of the scale-determining equation for advanced turbulence models," *AIAA journal*, Vol. 26, No. 11, 1988, pp. 1299–1310.

[28]Mavriplis, D. J., "Multigrid Strategies for Viscous Flow Solvers on Anisotropic Unstructured Meshes," *Journal of Computational Physics*, Vol. 145, No. 1, 1998, pp. 141–165.

[29]Mavriplis, D. and Venkatakrishnan, V., "A unified multigrid solver for the Navier-Stokes equations on mixed element meshes," *International Journal of Computational Fluid Dynamics*, Vol. 8, No. 4, 1997, pp. 247–263.

[30]Mavriplis, D. and Pirzadeh, S., *Large-scale parallel unstructured mesh computations for 3D high-lift analysis*, American Institute of Aeronautics and Astronautics, 2015/06/01 1999.

[31]Mavriplis, D. J., "Third Drag Prediction Workshop Results Using the NSU3D Unstructured Mesh Solver," *Journal of Aircraft*, Vol. 45, No. 3, 2015/06/01 2008, pp. 750–761.

[32]Yang, Z. and Mavriplis, D. J., "Higher-Order Time Integration Schemes for Aeroelastic Applications on Unstructured Meshes," *AIAA journal*, Vol. 45, No. 1, 2007, pp. 138–150.

[33]Mavriplis, D. J., Aftosmis, M. J., and Berger, M., "High Resolution Aerospace Applications using the NASA Columbia Supercomputer," *International Journal of High Performance Computing Applications*, Vol. 21, No. 1, 2007, pp. 106–126.

[34]Kirby, A. C., Mavriplis, D. J., and Wissink, A. M., "An Adaptive Explicit 3D Discontinuous Galerkin Solver for Unsteady Problems," 2015, AIAA Paper 2015-3046, 22nd AIAA Computational Fluid Dynamics Conference, Dallas, TX, June 2015.

[35]Gunney, B. T., Wissink, A. M., and Hysom, D. A., "Parallel clustering algorithms for structured AMR," *Journal of Parallel and Distributed Computing*, Vol. 66, No. 11, 2006, pp. 1419–1430.

[36]Anderson, R., Arrighi, W., Elliott, N., Gunney, B., and Hornung, R., "SAMRAI Concepts and Software Design," Feb 2013, `https://computation.llnl.gov/project/SAMRAI/download/SAMRAI-Concepts_SoftwareDesign.pdf`.

[37]Ringleb, F., "Exakte Loesungen der Differentialgleichungen einer adiabatischen Gasstroemung," *A. Angew. Math. Mech.*, Vol. 20, No. 4, 1940, pp. 185–198.

[38]Brachet, M., "Direct simulation of three-dimensional turbulence in the Taylor—Green vortex," *Fluid Dynamics Research*, Vol. 8, No. 1, 1991, pp. 1–8.

[39]National Center for Atmospheric Research, Boulder, CO, *Yellowstone: IBM iDataPlex System (Climate Simulation Laboratory)*, 2012, http://n2t.net/ark:/85065/d7wd3xhc.

[40]Bonet, J. and Peraire, J., "An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems," *International Journal for Numerical Methods in Engineering*, Vol. 31, No. 1, 1991, pp. 1–17.

[41]Fornberg, B., "Steady viscous flow past a sphere at high Reynolds numbers," *Journal of Fluid Mechanics*, Vol. 190, 1988, pp. 471–489.

[42]Fadlun, E., Verzicco, R., Orlandi, P., and Mohd-Yusof, J., "Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations," *Journal of Computational Physics*, Vol. 161, No. 1, 2000, pp. 35–60.

[43]Morrison, F. A., *An introduction to fluid mechanics*, Cambridge University Press, 2013.

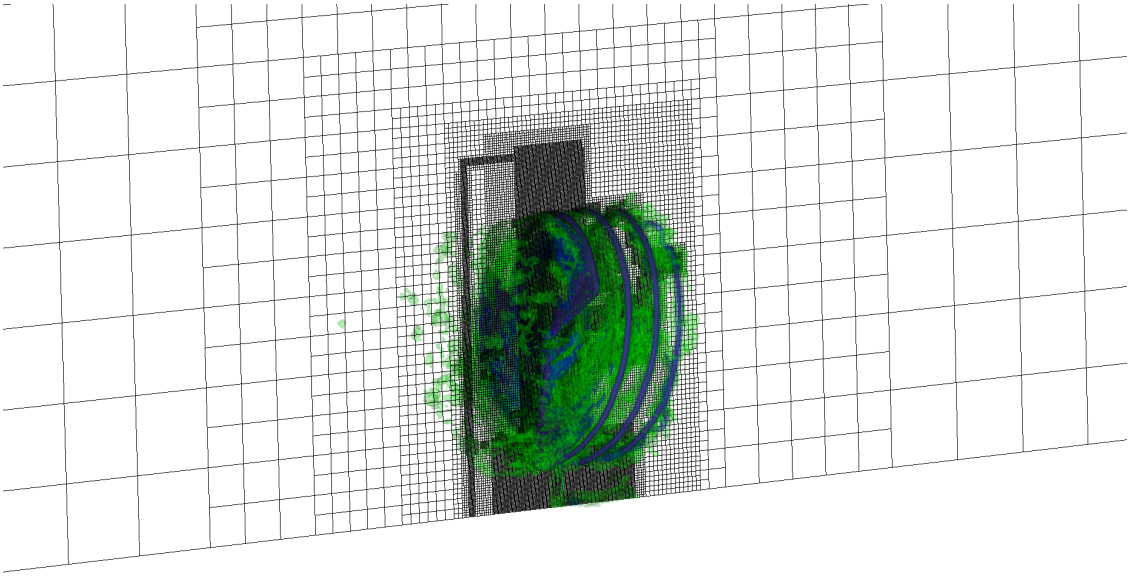
[44]McAlister, K. W. and Takahashi, R., "NACA 0015 wing pressure and trailing vortex measurements," Tech. rep., DTIC Document, 1991.

[45]Wissink, A., "An Overset Dual-Mesh Solver for Computational Fluid Dynamics," 2012, 7th International Conference on Computational Fluid Dynamics, Paper ICCFD7-1206, Hawaii.

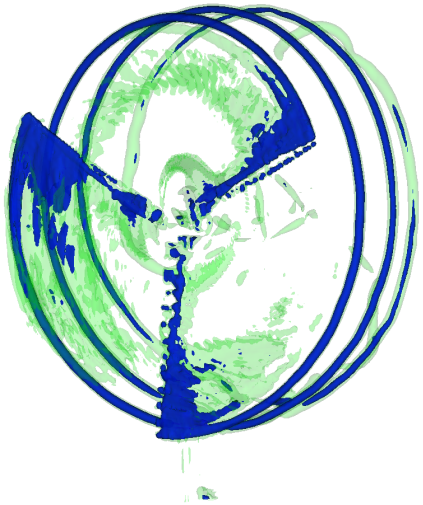
[46]Sitaraman, J. and Baeder, J. D., "Evaluation of the wake prediction methodologies used in CFD based rotor airload computations," 2006, AIAA Paper 2006-3472, AIAA 24th Conference on Applied Aerodynamics, Washington, DC.

[47]Hariharan, N., "Rotary-Wing Wake Capturing: High-Order Schemes Toward Minimizing Numerical Vortex Dissipation," *Journal of aircraft*, Vol. 39, No. 5, 2002, pp. 822–829.

[48]Jonkman, J. M., Butterfield, S., Musial, W., and Scott, G., *Definition of a 5-MW reference wind turbine for offshore system development*, NREL/TP-500-38060, National Renewable Energy Laboratory Golden, CO, 2009.



(a) Iso-contours of vorticity magnitude and a slice of the AMR grid.



(b) Close up view of iso-contours of vorticity magnitude.

(c) Close up side view of iso-contours of vorticity magnitude.

Figure 12: Iso-contours of vorticity magnitude for flow around a NREL 5MW wind turbine.