Computers & Fluids 92 (2014) 138–150

Contents lists available at ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

An efficient actuating blade model for unsteady rotating system wake simulations

C.E. Lynch [1], D.T. Prosser *,[1], M.J. Smith [1]

School of Aerospace Engineering, The Georgia Institute of Technology, Atlanta, GA 30332-0150, USA

ARTICLE INFO

Article history:
Received 19 March 2013
Received in revised form 3 December 2013
Accepted 10 December 2013
Available online 27 December 2013

Keywords:
Rotating system
Wind turbine
CFD
Actuator blade
Wake

ABSTRACT

This paper describes an innovative, efficient actuating blade model to capture the unsteady motion of a rotating system within Computational Fluid Dynamics (CFD) methods, with application to wind turbine blades. Each blade planform is modeled via a cloud of sources that move independently during the simulation to provide rotation of the blade as well as optional motion such as blade flexibility (aeroelasticity) and active controls (flaps, morphing, adaptive shapes). The model can be implemented into structured or unstructured methods that span the gamut from full potential to Large Eddy Simulations (LES), and it does not require the use of overset grids. A key feature of this model is the development of a highly efficient parallelized kd-tree algorithm to determine the interactions between actuator sources and grid nodes. Computational evaluation of the method successfully demonstrates its capability to predict root and tip vortex location and strength compared to an overset Navier–Stokes methodology on an identical background grid, and further improvements in the solution are shown by the use of grid adaptation.

1. Introduction

Rotating systems are abundant in engineering fluid dynamics applications. A few examples include wind turbines, propellers, rotors, compressors, and jet engines. Wind turbines represent one type of rotating system that has gained incredible interest in recent years due to the push towards alternative energy sources. Despite their relatively simple appearance, horizontal-axis wind turbines (HAWTs) operate in an aerodynamic environment that is challenging to model. Due to the atmospheric boundary layer, there can be considerable variation in wind speed between the top and bottom of the rotor disc. If the turbine is not facing directly into the wind, there will be yaw error and the blade loads will vary cyclically as they rotate. All wind turbines have some kind of yaw control, but the wind direction can vary too quickly for the controller to maintain zero yaw. At high wind speeds, even in axial flow, the blades may be stalled.

Though Computational Fluid Dynamics (CFD) has made significant inroads as a research tool in wind turbine aerodynamics [1–3], simple, inexpensive methods are still the workhorses in design and aeroelasticity applications [4]. These can range from blade element momentum (BEM) theory methods [5] to more accurate but still inexpensive vortex methods [6]. BEM methods provide basic insight into turbine flows, but only under the simplest conditions: constant wind speed with zero yaw error. BEM methods operate under the *independence principle*, in which the aerodynamics of each airfoil section along the blades are computed independently of neighboring sections [7]. As a result BEM methods completely neglect spanwise flow and other three dimensional (3-D) effects, which have been shown to result in significant lift augmentation and stall delay, especially near the blade roots [8]. Vortex methods such as prescribed wake models [9] can capture some unsteady effects, but like BEM methods, lack the ability to handle 3-D effects. As a result, both typically underpredict torque, even when they incorporate a 3-D correction [10]. Designs based on such simulations can result in structures that succumb to fatigue sooner that expected [11,12].

CFD techniques can mitigate many of the inaccuracies from the simplifying assumptions for wind turbine analysis methods. There are currently four broad classes of recent CFD improvements that can be applied to improve wind turbine design and analysis tools. The first class is based on hybrid Reynolds-Averaged Navier–Stokes (RANS)–Large Eddy Simulation (LES) methods, where improved turbulence models can improve the prediction of unsteady, separated flows. RANS and LES usually apply the concept of overset grids, which introduce the ability to treat the relative motion between rotor and its support structure. Source-based methods, including actuator disks and actuator lines, can provide physics-based characterizations of wind turbine wakes while reducing computational expense associated with modeling the blades, and do not require overset grids. As wind turbine blades continue to

* Corresponding author.
E-mail address: dtprosser@gatech.edu (D.T. Prosser).

[1] Research is supported by the National Research Foundation under Grant number CBET-0731034.

0045-7930/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compfluid.2013.12.014

increase in size, coupling between CFD and computational structural dynamics (CSD) methodologies to capture the aeroelastic response of the rotor blades becomes increasingly important.

The momentum source technique is one approach to managing complexity and computational expense in simulations of wind turbines. Actuator disc methods, based on momentum theory, seek to model the *effects* of a rotor on the surrounding flowfield without the need to model the physical rotor. Since the flow over the rotor blades produces lift and drag forces on the blades, there must be another force equal in magnitude and opposite in direction acting on the flow. This reaction force can be included in the underlying CFD model in two ways. It can be implemented in a manner similar to a boundary condition. A special permeable boundary surface is embedded in the mesh, with the flow velocity constrained to be continuous through the surface while the pressure is discontinuous. Alternatively, those forces can be included as body forces in the Navier–Stokes equations. The body force approach offers additional flexibility over the pressure discontinuity. Since no surface is needed in the grid, the position of the disc can change, allowing different choices of the tip path plane on the same grid. In either case, the blades themselves are neglected. The pressure discontinuity or body forces provide the same effects on the flowfield as rotating blades, but in a time-averaged sense. Without modeling the blades directly, CFD simulations can either use fewer grid points, speeding their execution compared to more detailed methods, or place more points in the turbine wake gaining greater modeling fidelity.

CFD with actuator discs has seen widespread use in both the rotorcraft and wind industries [13–16]. For a good review of actuator disc techniques, largely from the perspective of structured-grid CFD, see Ref. [17]. O'Brien implemented unsteady actuator disc and blade techniques in the unstructured solver FUN3D and compared those against fully overset techniques on several geometries, demonstrating that an actuator disc can significantly improve predictions of helicopter fuselage loads [18,19]. While actuator disc methods provide a good approximation of the influence of the rotor, the fact that they lack discrete blades means that they model the rotor in an azimuthally averaged fashion. This makes them unsuitable for cases with yaw error since the rotor wake varies azimuthally, as well as radially. O'Brien demonstrated that an actuator disc acts as a elliptic wing and generates the roll-up of two tip vortices in the near wake, but they cannot capture the helical vortex wake associated with rotating blade systems, which an actuator-blade method was able to predict [19].

To address this issue for wind turbines, Sørensen et al. applied a variant of the actuator-blade method, known as an actuator-line method, whereby the blades are modeled by filaments or lines of sources along which body forces act [20,21]. These body forces are typically derived from a BEM method that uses tabulated airfoil data. In this approach, the unsteady effect on the flowfield by individual blades can be included in the analysis while still avoiding direct CFD modeling of the blade surfaces and their associated boundary layers. Sørensen et al. applied this actuator line method to a 500 kW Nordtank turbine and achieved good agreement with the experimental power curve for pre-stall wind speeds. Above about 12 m/s, where that particular rotor is stalled, they over-predicted power because using airfoil data implicitly ignores the three-dimensional effects present in the stalled regime.

The actuator line method has also been applied with good results by Mikkelsen, who compared it against an axisymmetric actuator disc approach and used it to validate some of the fundamental assumptions of traditional BEM methods [22]. Whereas Sørensen et al. used a code that solves the Navier–Stokes equations in vorticity–velocity form, Mikkelsen employed the incompressible code EllipSys3D [23,24], which solves in pressure–velocity form. This latter method permits the inclusion of solid boundaries like the turbine tower.

The EllipSys3D code was also used by Ivanell et al., who evaluated the vortical wake structures produced by the actuator line method [25]. In that work, the actuator lines were fixed in the grid, and the effects of blade rotation were applied via a boundary condition, which allowed them to use an efficient steady-state formulation. While this significantly decreases computational expense, it renders the method unsuitable for yawed cases. In addition, the azimuthal boundary condition was also periodic, so a N-bladed rotor could be modeled in only $1/N$ revolutions. Again, this precludes the application of the method to any case that does not have a periodic solution, which are the cases primary configurations of interest.

O'Brien's actuator blade method was originally applied to helicopter rotor-fuselage interaction problems using an unstructured CFD code [18,19]. The key difference between actuator line and actuator blade methods is that in the former, each blade consists of a single line of sources. To avoid discontinuities, each source's loading is distributed over multiple grid points using a "regularization function" that makes a source's influence at a distance r away from it scale with e^{r^2}. Conversely, the actuator blade method of O'Brien uses a rectangular array of sources for each blade, providing a continuous influence without the need for a regularization function (though smaller discontinuities in loading do still exist at the outlines of the blade). A rectangular array of sources also provides the means to vary the local angle of attack (a key input to the underlying blade element model) with chord. It should be noted, however, that the results presented here use a local angle of attack that is constant along the chord but still varying along the span.

Though they are undoubtably easier to apply to complex geometries, source cloud actuator methods like that of O'Brien [19] suffer from one significant drawback. In order to apply body forces from the actuator sources to the flow, it is necessary to know which grid node (or cell centroid for a cell-centered code) is closest to each actuator source. This entails some sort of nearest neighbor search procedure. In actuator disc cases or actuator line/blade cases in which the sources are fixed in the grid (perhaps because the equations are being solved in a rotating frame), this search can be performed as a one-time pre-processing step. In this instance, the cost of the search algorithm is not of paramount importance. However, if the sources move in relation to the inertial grid, the search must be repeated at each time step.

In a structured grid, there is a regular structure in memory that mimics planar spatial structures, making it readily efficient to search through a range of grid indices. In an unstructured grid, a much more general search procedure is required. With the exception of O'Brien's work, the actuator methods discussed thus far have been implemented within structured CFD methods. O'Brien's implementation for unstructured grids performs an exhaustive nearest-neighbor search by looping over all the grid nodes to find the node associated with a single source, and then repeating that loop for all the other sources. The exhaustive search represents another significant expense in addition to the usual computational overhead associated with storing and accessing connectivity information in an unstructured grid, so a better search algorithm is necessary for practical engineering applications.

In this paper, an efficient method for modeling rotating systems in engineering fluid dynamics applications is presented. The method is demonstrated using a wind turbine model for validations. This paper documents the adaption of the actuator blade model using clouds of sources that relocate in an inertial background grid that can include other stationary objects such towers and other turbines (with or without moving blades). In particular, details of a parallel search algorithm are included, as this implementation

is mandatory to ensure the cost-effectiveness of the method is comparable to other source-based approaches. Performance and wake correlation with a full overset CFD simulation on an identical background grid are provided to demonstrate the ability of the algorithm. Finally, grid adaptation is performed to show how the wake preservation can be improved in an efficient manner using this modeling framework.

2. FUN3D code

FUN3D is NASA Langley's code for solving the Navier–Stokes equations on mixed element unstructured grids [26,27]. It can solve both compressible and incompressible flows, the latter achieved via Chorin's method of artificial compressibility [28]. The spatial solution is resolved via an implicit node-centered finite-volume formulation. Steady flows are solved using a first order backward Euler scheme, while time accurate simulations are integrated with a second order backward differentiation formula (BDF). By default, the inviscid fluxes are evaluated using Roe upwinding. The viscous fluxes are always evaluated using a scheme that is equivalent to a central difference formulation. The resulting linear system is solved using a point-implicit red–black Gauss–Seidel scheme. A number of turbulence methods are available, including Spalart–Allmaras, Menter kω-SST and Detached Eddy Simulation (DES) models.

The overset grid option can be exercised through the DiRTlib (Donor Receptor Transaction Library) [29] and SUGGAR (Structured, Unstructured, and Generalized Grid AssembleR) [30] libraries. With structured grids, overset methods provide a means for handling geometric complexity. Since the unstructured grids in FUN3D are already amenable to complex geometries, overset methods are primarily used to resolve moving body problems. FUN3D has been applied and validated for a very wide range of moving body applications, including rotorcraft and wind turbines [19,31–33].

3. Actuator blade algorithm

O'Brien demonstrated that an actuator blade methodology can provide a reasonable approximation of the unsteady loads on a helicopter fuselage due to the rotor wake [19,34]. As discussed earlier, these methods model the influence of the rotor on the flowfield by adding source terms to the Navier–Stokes equations that act as body forces. The local lift is estimated using the linear estimate of the lift coefficient:

$$C_l(\alpha) = C_{l_\alpha}(\alpha - \alpha_0) + C_{l_0} \tag{1}$$

which was shown by O'Brien to be sufficient for the actuator methodologies. If desired, the source array representation of the blade could be used to vary the local value of α along the chord in order to model morphing blades or flaps for active control, for example. In this study, however, α is represented as constant along the chord. The lift curve slope as well as parameters such as maximum and minimum lift coefficient can be input into the solver in order to match the characteristics of the rotor blade being used. Unlike actuator line methods, the actuator blade does not require averaging of the source data to prevent singularities. Inflow velocity is taken from the grid node nearest to each source on the leading edge of each actuator blade. The twist of the blade can be set using a linear variation or interpolated by the solver from a table lookup file. Taking into account the inflow velocity and the blade twist, the effective angle of attack is computed at each radial location. The local lift coefficient is computed at each spanwise source, and the lift and drag forces are determined using Eq. (2) at each successive radial location. The sectional lift and drag are then distributed evenly over all the sources at each spanwise location, so that there is no variation of these parameters in the chordwise direction. However, there is variation in the radial direction because the local angle of attack varies radially.

$$\begin{aligned} \Delta L' &= 1/2 \cdot C_l \cdot \rho \cdot V_{Total}^2 \cdot c \cdot \Delta r \\ \Delta D' &= 1/2 \cdot C_d \cdot \rho \cdot V_{Total}^2 \cdot c \cdot \Delta r \end{aligned} \tag{2}$$

The actuator blade portion of one time step of the solution begins by computing the new positions of the actuator sources. These sources rotate through the mesh in clouds that reproduce the approximate blade planform as shown in Fig. 1. Then a search is done to find the nearest neighboring mesh vertex to each source. Since each processor stores *all* of the actuator sources, but only a subset of the mesh, a source may actually lie outside the mesh partition. In that case, the source's true nearest neighbor actually lies on another partition. To deal with that situation, the processors exchange the distance they computed from each source to its nearest vertex. If another processor has a smaller distance, that source is marked as belonging to another partition, and its influence is not included on that processor. This "multiple partition check" can be handled with an efficient `MPI_Reduce` operation. Following that check, the state variables at the vertex associated with each source are used to computed the local effective angle of attack. That angle of attack is then applied in a blade element model to compute source strengths. Finally those source strengths are added to the residual and Jacobian, and the linear system is solved. A more complete discussion of this overall process can be found in Ref. [19].

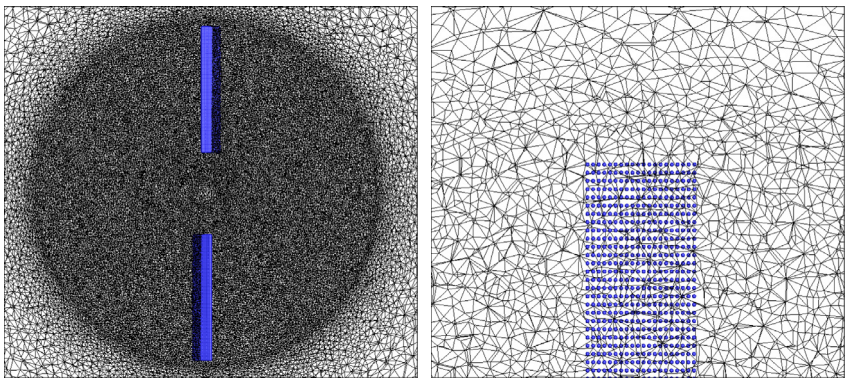
The baseline actuator blade implementation [34] resolved the source-to-node association through a nearest neighbor search. Originally, a straightforward exhaustive search was applied to demonstrate the concept. This method computes the distance between each source and each mesh vertex, resulting in an operation count of $O(N_v N_s)$, where N_s is the number of actuator sources, and N_v is the number of mesh vertices resident on a particular processor. In a typical case, N_v might range from 30,000 to 200,000, and N_s will be on the order of 2000 sources per blade. Thus, for some configurations, this search can become a very expensive operation. Indeed, O'Brien found that the cost of an actuator blade solution using this implementation can quickly approach the cost of a solution with overset blades [34], rendering it not useful for engineering applications. Thus, a new search algorithm was necessary, as is discussed in Section 4.

4. Kd-tree search algorithms

Nearest neighbor searches have been thoroughly studied in the computer science literature, and many search algorithms exist that are faster than the exhaustive search described above, at least in three dimensions. *Space-partitioning* methods are a class of algorithms in which the set of points being searched is recursively divided into smaller subsets. The divided set may be represented by a tree-based data structure such as an *octree* or a *kd-tree*. The latter is known to be one of the best data structures for nearest neighbor searches in two or three dimensions [35]. In general, a kd-tree nearest neighbor query has $O(\log N)$ complexity, where N is the number of nodes in the tree [36].

4.1. Tree construction and nearest neighbor queries

For purposes of actuator source-to-node association, a node of a kd-tree contains a list of at most m_v mesh vertices, including the vertices' array indices and their positions in space, as well as pointers to left and right child nodes. Each tree node also has a split direction, d, and a split point, $\vec{s}$. All vertices in the left child have



(a) Complete rotor (b) Close-up of tip

Fig. 1. Actuator sources (blue) embedded in mesh. Each source must be associated with the nearest mesh vertex. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$p_d \leqslant s_d$, and all vertices in the right child have $p_d > s_d$, where $\vec{p}$ is a vertex's position in space.

Kd-trees are constructed recursively, so the construction procedure proceeds identically at any depth in the tree. The process begins with a node being passed a list of n_v mesh vertices (at the beginning, all vertices are passed to the root node). If $n_v \leqslant m_v$, all of the vertices are inserted into the node's list, and control returns to the parent node. If $n_v > m_v$, the vertices are sorted by their position in the split direction, d. Once sorted, the lower half of the vertices are sent to the left child, and the upper half are sent to the right child. The median point alone is inserted into the current node. The split direction alternates through the tree, so that if a node is split in direction, d, its two children are split in the `mod(d,3) + 1` direction (assuming 1-based indexing).

To demonstrate this concept, consider the following list of six 2-D vertices: $(4,7)$, $(9,6)$, $(5,4)$, $(2,3)$, $(7,2)$, $(8,1)$. Assume $m_v = 1$, meaning that only one vertex may be stored in each tree node, and that the initial split direction is x. First, the list is sorted in the x-direction. The median of the list is $(7,2)$ and is inserted into the root node of the tree (Fig. 2a). That vertex becomes the split point for the root node. The nodes to the left of $(7,2)$ are now sorted in the y-direction. The median of that sublist is $(5,4)$, which is inserted into the left child of the root (Fig. 2b). The only vertex to the left of (below) $(5,4)$ is $(2,3)$, so it goes into the left child of the root's left child (Fig. 2c). Vertex $(4,7)$ is handled similarly (Fig. 2d). Then control moves to the right half of the original sublist: $(8,1)$ and $(9,6)$. That sublist is sorted in the y-direction. The median is $(9,6)$, which is inserted into the right child of the root, and then that child is split at that point (Fig. 2e). The only remaining node is inserted into the left (bottom) child of the root's right child (Fig. 2f).

Searching a kd-tree for the vertex nearest to a query point $\vec{q}$ can begin at any node, though typically it begins at the root node. The number of points stored in a node and its children is n_v. If the node has $n_v \leqslant m_v$, it is a leaf, and a simple exhaustive search is done over its vertices. That is, the distance squared between q and p is computed for each vertex and then compared to the smallest distance found so far. If it is smaller, it becomes the new "best" distance, and that vertex's index is saved. Control then returns to the parent node since leaf nodes have no children.

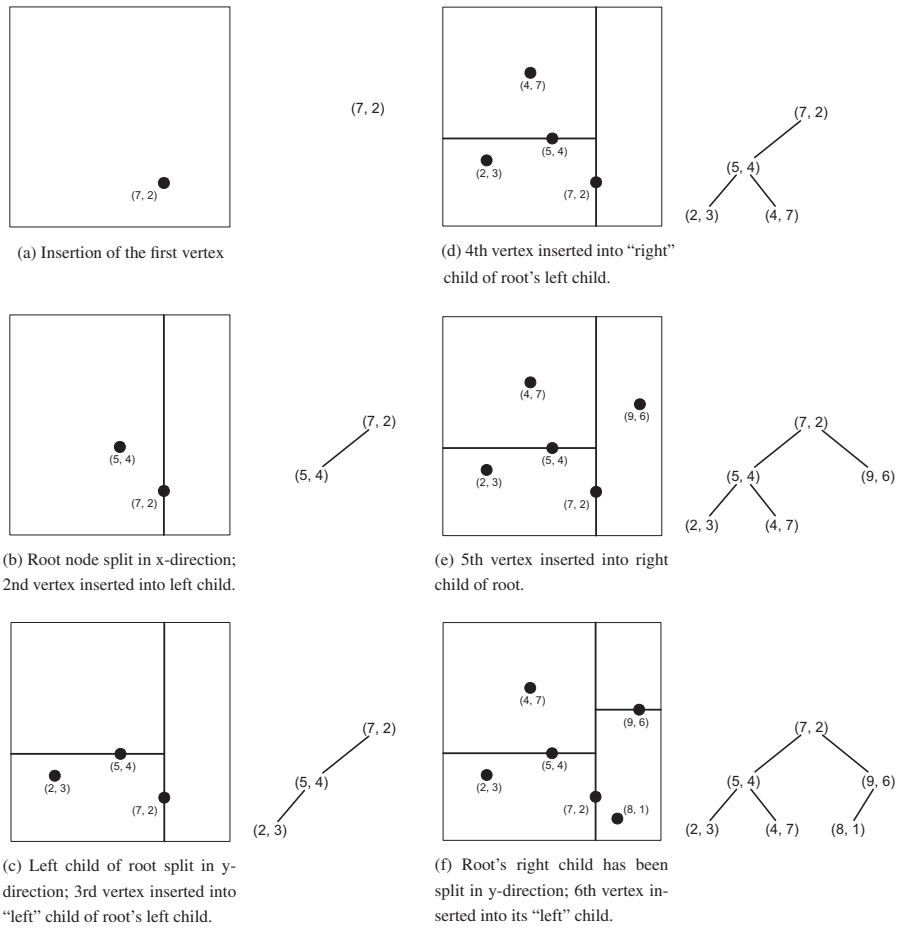
If the node has $n_v > m_v$, the distance squared between $\vec{q}$ and the one point actually stored in the node is computed and compared to the minimum distance encountered so far. If the query point lies to the left of the node's split point ($q_d < s_d$), the left child is recursively searched. Otherwise, the right child is searched. After that search, the distance from the query point to the plane separating the two children is computed. If it is smaller than the minimum distance encountered so far, it is possible that the nearest vertex to the query point lies just across that plane in the child that was not searched previously. In that case the other child is searched recursively as well. Finally, control is returned to the parent node.

Consider the example tree constructed earlier and a query point $(1,5)$. A search for the query point's nearest neighbor begins at the root node with vertex $(7,2)$. The distance squared from the query point to that vertex is 45 (Fig. 3a). Since the query point is to the left of the root node's split point, the search moves to the root's left child, which contains the vertex $(5,4)$. The distance to that vertex is 17, which is the minimum value encountered so far (Fig. 3b). Since the query point is to the "right" of (above) the split point, the right child with vertex $(4,7)$ is next examined. That distance is 13, which now becomes the closest vertex (Fig. 3c). Since this node has no children, the search computes the distance to the plane separating it from its sibling. The distance to the plane is 1, which is less than the distance to the best vertex found so far (Fig. 3d). Therefore, a vertex lying just on the other side of the plane could be a nearer neighbor, and the search evaluates the sibling. The vertex contained in that sibling has a distance of 5 from the query point, and so becomes the best candidate (Fig. 3e). Now control returns to the parent node, with vertex $(5,4)$. Both its children have been searched, so control returns to the root node. The distance from the query point to the root node's split plane is 36, which is greater than the distance for the best candidate, and so the right child of the root node need not be searched (Fig. 3f). The nearest neighbor of the query point is $(2,3)$. Since the search never descended into the right tree, the expensive distance function was never evaluated for those vertices.

4.2. Kd-tree performance evaluation

Performance of a kd-tree search will depend on several factors. In order of importance, they are the total number of vertices contained in the tree, N_v; the maximum number of vertices contained in any one node, m_v; and the efficiency with which the distance function is evaluated. The third factor is addressed by always using squared distances so as to avoid computing square roots and by computing distances inline without a separate function call.

To determine how the search algorithm scales with the maximum number of vertices allowed per tree node, a simple scaling study was done using four central processing units (CPU) on a mesh with 154,082 vertices, varying m_v. Altering m_v essentially changes the number of nodes in the kd-tree. Since kd-tree searches scale logarithmically with the number of nodes, m_v was changed by powers of two. Fig. 4 shows the ratio of the average cost of a kd-tree search to an exhaustive search vs. m_v. In this test, the best



(a) Insertion of the first vertex

(b) Root node split in x-direction; 2nd vertex inserted into left child.

(c) Left child of root split in y-direction; 3rd vertex inserted into "left" child of root's left child.

(d) 4th vertex inserted into "right" child of root's left child.

(e) 5th vertex inserted into right child of root.

(f) Root's right child has been split in y-direction; 6th vertex inserted into its "left" child.

Fig. 2. Graphical depiction of the algorithm for building an $m_v = 1$ kd-tree. The partitioning of physical space is shown on the left, and the data structure is shown on the right.

performance was obtained with m_v set to either 64 or 128 (the difference between the two being negligible).

To determine the importance of N_v and m_v, several actuator blade simulations were undertaken on two similar grids. The grids contained about 2.4 and 4.5 million vertices, respectively. Both grids contained no solid boundaries. Mesh points were clustered in a region between the rotor disc and another disc about three rotor diameters downstream. The number of vertices in each CPU's tree was altered by varying the number of CPUs used in each simulation. The number of vertices per processor ranged from 17,651 to 282,420. For each of those node counts, simulations were conducted using the pre-existing exhaustive search; a kd-tree search with $m_v = 128$, previously identified as optimal; and a kd-tree search with $m_v = 16$, a reasonable but still sub-optimal value. The average search time, as well as the average total time (including activities not related to actuator blades) per time step for the kd-tree was compared to the exhaustive search.

The kd-tree to exhaustive search cost ratios are shown in Fig. 5. Clearly, when the CPUs are very heavily loaded with a large number of vertices, the kd-tree is vastly superior, with less than 15% the search cost of the exhaustive search (solid lines). There are diminishing returns from the kd-tree as the number of vertices per CPU decreases (i.e., as the number of CPUs increases, since the total number of vertices is fixed). On the larger mesh (grid 2), the kd-tree is 88% as expensive as the exhaustive search. Also, at low numbers of vertices, the search procedure is overall a very small percentage of the total time per time step. This occurs because the exhaustive search makes very efficient use of cache and avoids the overhead of function calls.

In a Fortran implementation of the present search algorithm, the memory usage for one kd-tree node is always:

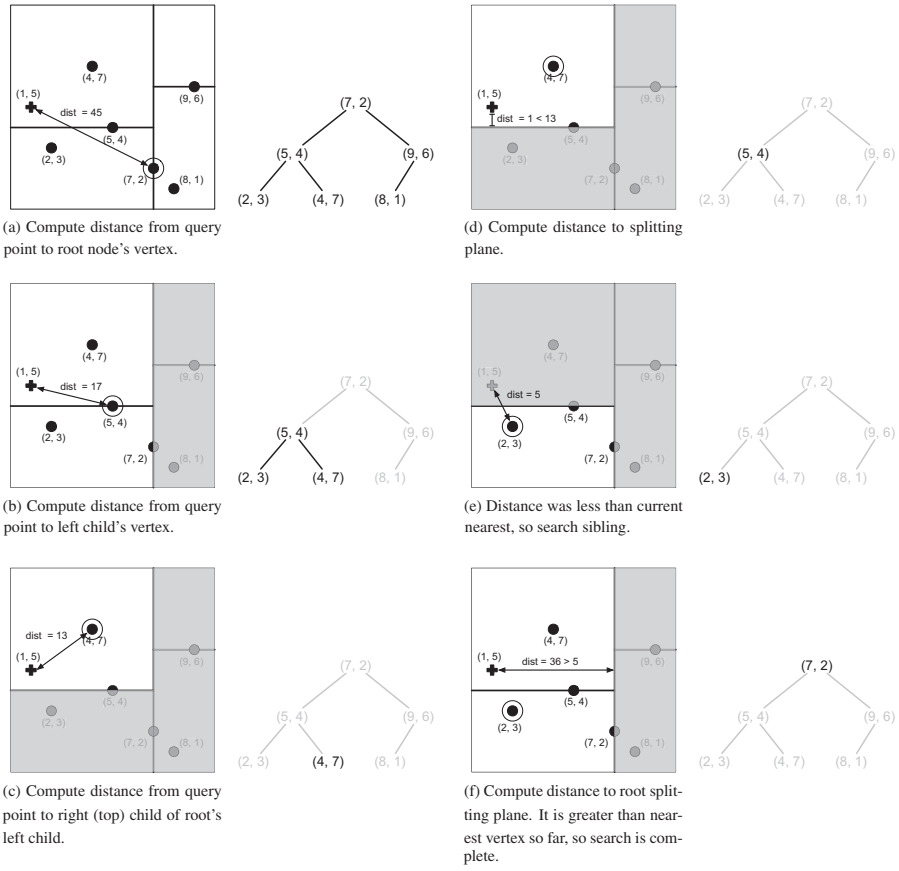
$$\begin{aligned}\text{memory per node} &= 4 + 4 + 4m_v + 8km_v + 8 + 8 \\ &= 24 + (4 + 8k)m_v \end{aligned} \tag{3}$$

In all applications considered here, $k = 3$. In a best-case scenario for memory usage, each leaf node would hold exactly m_v vertices, minimizing the total number of nodes. In a worst-case scenario, each would hold only one vertex, giving the largest possible number of nodes and essentially wasting $28(m_v - 1)$ bytes per node. So memory usage will be in the range:

$$\frac{N_v}{m_v}[24 + 28m_v] < \text{total memory} < N_v[24 + 28m_v]\,\text{bytes} \tag{4}$$

Currently, memory usage is within acceptable bounds. Should it become necessary to reduce memory usage, the algorithm could be modified to directly index the arrays holding the mesh vertices rather than copying them. This would save $24m_v$ bytes per node.

The effect of m_v, the maximum number of vertices per tree node, is striking. Allowing more vertices per tree node improves performance of the kd-tree considerably, as can be seen in the distance between circles and triangles in Fig. 5. Increasing m_v allows the kd-tree algorithm to gain back some of the cache efficiency of an exhaustive search. Furthermore, it reduces the maximum depth of the tree so that fewer recursive function calls are required to reach a leaf node. Indeed, as can be seen on the smaller of the two grids (Fig. 5a), using 128 vertices per node rather than 16



(a) Compute distance from query point to root node's vertex.

(b) Compute distance from query point to left child's vertex.

(c) Compute distance from query point to right (top) child of root's left child.

(d) Compute distance to splitting plane.

(e) Distance was less than current nearest, so search sibling.

(f) Compute distance to root splitting plane. It is greater than nearest vertex so far, so search is complete.

Fig. 3. Graphical depiction of the algorithm for search a kd-tree with $m_v = 1$ (continued). Vertices are represented by black dots, the query point by a black cross, and the current nearest neighbor by an unfilled circle. Gray regions are not currently being searched.

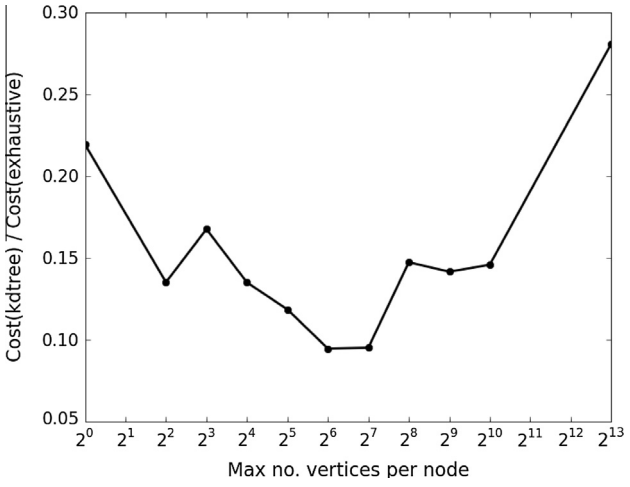


Fig. 4. Ratio of the cost of a kd-tree search to the cost of an exhaustive search vs. the maximum number of vertices per CPU.

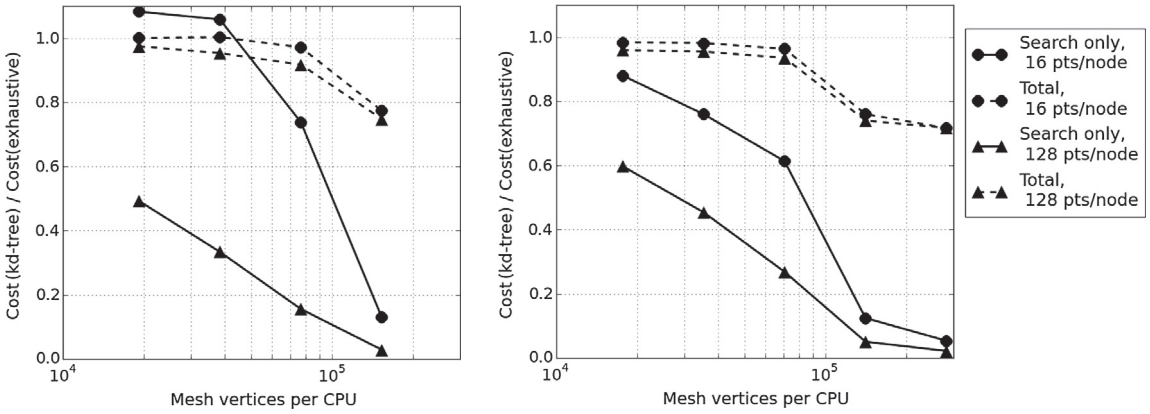
ensures that the kd-tree search is always less expensive than exhaustive search.

5. Wind turbine wake prediction

To demonstrate the capability of the actuator blade approach for a rotating system, the NREL Phase VI wind turbine [37] was chosen. A wind speed of 7 m/s was chosen since at higher speeds the flow over the blades begins to stall. Obviously, if a blade element module is used to determine the source strength, the accuracy of the actuator blade method will diminish when the flow is dominated by viscous effects near solid boundaries or other phenomena, like stall, that depend greatly on the details of the geometry involved. More complex methods that include stall models can be applied in these situations. This approach and test cases were chosen so that the method could be demonstrated without the addition of uncertainties due to dynamic stall and other nonlinearities, which are dependent on empirical stall models. It is emphasized that the model and approach are not limited to the linear regime. The Reynolds number, based on the rotor radius of 5 m and the tip speed of 37.7 m/s, was 1.3×10^6.

The incompressible path in FUN3D was chosen to resolve the flow field as the freestream and tip Mach numbers are very low, less than 0.1. Since no solid boundaries are present in the CFD calculation, the wall distance function used in most turbulence models cannot be evaluated, and so only laminar simulations were performed. Current efforts involve modifying the distance function to treat actuator sources as surface nodes for the purposes of calculating wall distance.

At each time step, source strengths were computed using a very simple linear model, as described in Ref. [19]. Linear lift curve data were estimated from S809 airfoil data in Ref. [37]. In particular, the lift curve slope was estimated at 6.54/radian, and the zero-lift angle of attack was estimated as 0.614°. Lift is limited to minimum and maximum values of −0.8 and 1.03 when the effective angle of attack is outside the linear region. For this demonstration, constant chord blades were demonstrated, so the mid-span chord of 0.5475 m was used. It should be noted that the method is not limited to constant chord blades.



(a) Grid 1, with 2.4 million vertices

(b) Grid 2, with 4.5 million vertices

Fig. 5. Ratio of the cost of the kd-tree search to the cost of the exhaustive search. Solid lines indicate the cost ratio for the search portion of a time step alone. Dashed lines indicate the cost ratio for a complete time step.

The actual twist distribution of the wind turbine blade, as supplied in Ref. [37], was used. Fig. 6 shows the local twist extracted from slices of the overset blade grid in comparison with that input for the actuator blade simulation, confirming that the two are consistent. Each actuator blade was represented by 2000 sources, 100 in the spanwise direction and 20 in the chordwise direction. This results in a source distribution finer than the mesh in the vicinity of the rotor, as shown in Fig. 1, ensuring that there are no mesh vertices in the region currently occupied by a blade that do not receive a body force from a source.

An overset CFD simulation with two blades but without a tower provided a comparison of the wake approximated by actuator sources to the wake from actual rotating blades. The mesh for the actuator blade simulation had a total of about 2.4 million nodes, with a cell size of 0.05 m (about 13% tip chord) near the hub. The mesh extended 8 rotor radii upstream of the hub, 10 radii downstream, and 5 radii in the spanwise direction. This mesh also served as the background mesh for the overset simulation. The overset blade meshes had about 2.7 million nodes each, resulting in a composite mesh of 7.8 million nodes. Surface spacing at the tip was about 0.006 m in all directions (an approximately isotropic surface mesh). Normal spacing of the grid was computed such that $y^+ < 1$, which is the recommended practice for turbulent flows. For ease of comparison with actuator cases, the overset simulations presented here are laminar. In engineering applications with overset grids, a turbulence model would be used, and so the turbulent grid generation conventions were applied for the overset grid. On the other hand, the source method is not constrained by highly refined grid spacing near the blade. It is demonstrated later in this paper that the tight spacing (and greater degrees of freedom) required to resolve boundary layers significantly increases computational expense compared to the actuator simulations.

In the overset and actuator cases, a time step equivalent to 1° of azimuth per step was used, which has been observed for a number of rotating system applications (rotorcraft) to be sufficient to capture the aerodynamic behavior of the rotor exclusive of dynamic stall [38]. A temporal error controller was used to ensure that enough Newton subiterations were used in each time step to reduce flow solver residuals to 10% of an estimate of the temporal error [39]. Fig. 7 shows the convergence of the x-momentum (R_2) and y-momentum (R_4) residuals plotted on a semi-log scale. Both simulations progressed for four rotor revolutions, after which the integrated torque was observed to be periodic. The wakes were stable without a turbulence model.

Fig. 8a and b shows a comparison of the vortical wake computed by the actuator blade and overset methods. The isosurfaces are defined by Q, which is given by

$$Q = \frac{1}{2}\left(||\Omega||^2 - ||S||^2\right) \tag{5}$$

where Ω is the vorticity tensor, and S is the rate-of-strain tensor. Away from the blades, the overset and actuator solutions are quite similar. The tip vortices predicted by the actuator blades are indistinguishable from those predicted by overset blades. The overall

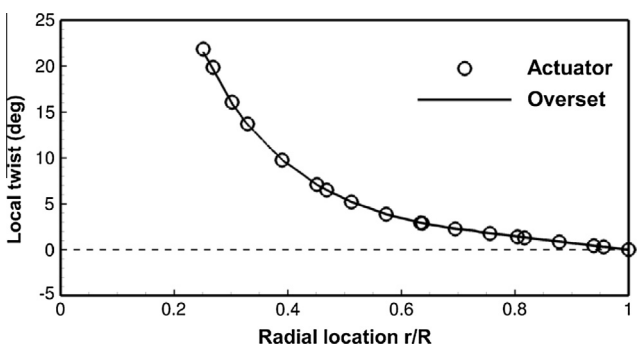


Fig. 6. Blade twist distribution for the overset blades and actuator blades.

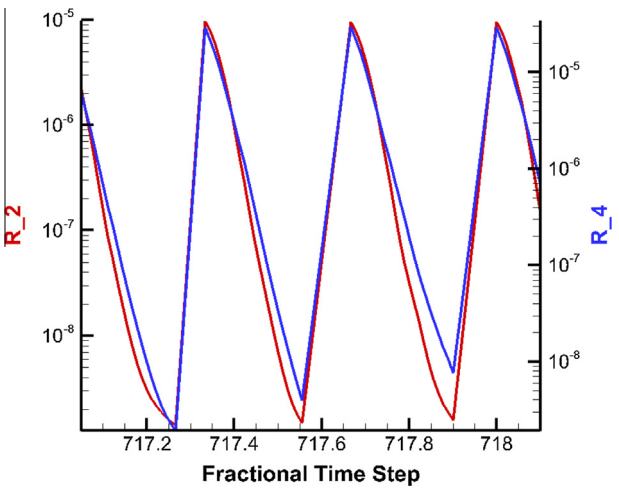
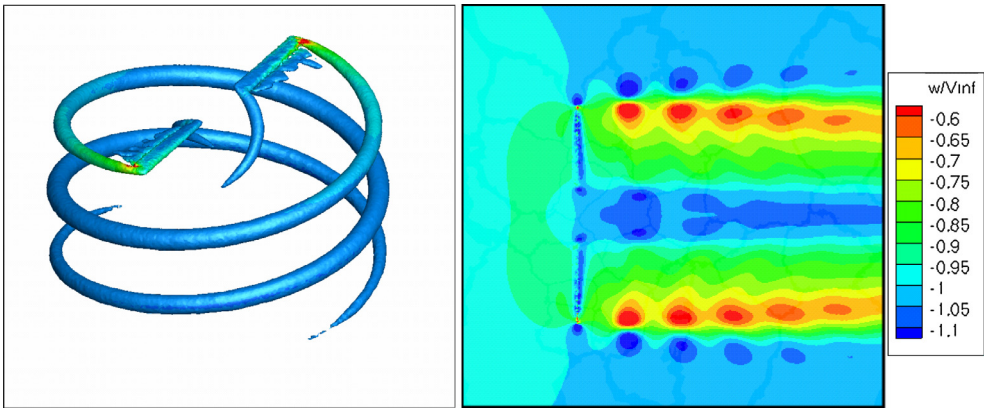
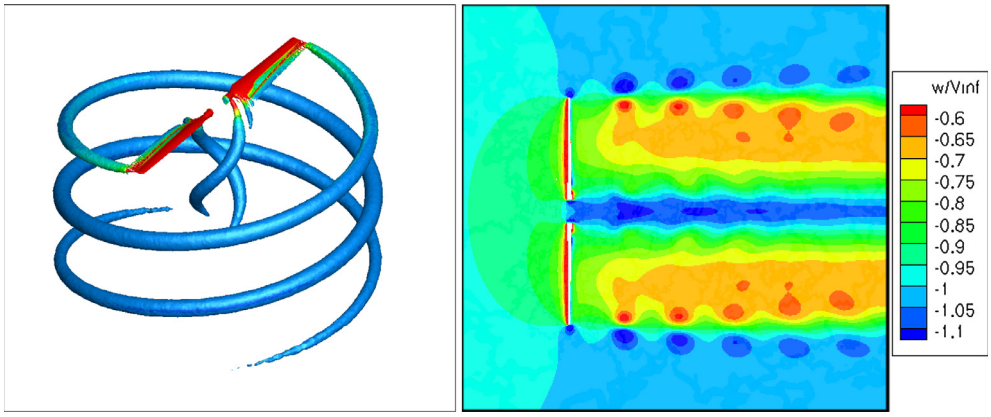


Fig. 7. x-Momentum and y-momentum residuals showing convergence via the temporal error controller.



(a) Actuator blades



(b) Overset blades

Fig. 8. $Q = 0.05$ iso-surfaces and contours of axial velocity, w/V_∞, along a centerline plane at $V_\infty = 7$ m/s, zero yaw. Q is normalized by R and U_{tip}. Both cases were run for four revolutions at 1° of azimuth per step.

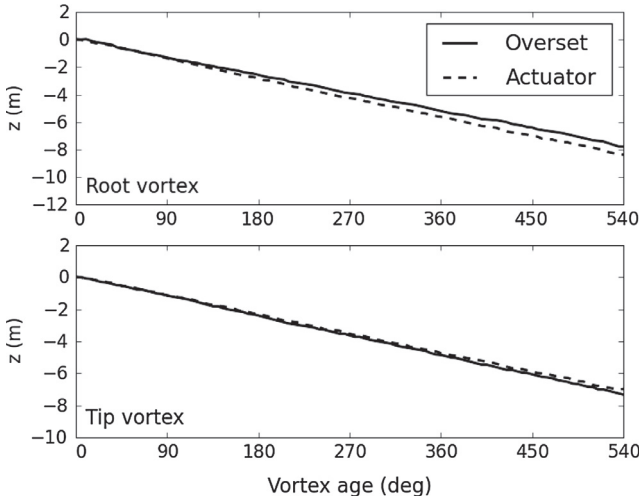
wake deficit is also well-captured. The "jet" of higher speed flow through the center of the disk is somewhat more diffuse in the actuator solution. This can be attributed to differing geometry near the hub. The inboard edge of the actuator blades corresponds to the first radial station with an airfoil profile, 1.257 m from the axis of rotation. The actual blades have a cylindrical root attachment starting 0.508 m from the hub followed by a gradual transition to the first airfoil section at 1.257 m. This simplification was necessary because the underlying blade element model of the actuator method assumes a linear airfoil-like relationship between angle of attack and lift, which will not hold for a cylindrical section. In contrast, the overset blades match the actual NREL model blades. In addition to being closer to the hub, the cylindrical root sheds a more powerful vortex than the actuator blades.

The trajectories of the vortices after leaving the blade root and tip provide another useful comparison of the actuator and overset blade methods. Fig. 9 shows the axial and radial components of those trajectories. From the axial component, it is clear that the actuator and overset blade methods convect tip vortices downstream at nearly the same speed. Conversely, the actuator blade method convects the root vortices faster. This is qualitatively evident from the contour plots of w/V_∞ in Fig. 8, where the actuator blade contour lines show a larger high speed core. This is due in part to the difference in the root-hub model of the overset and actuator blade configuration. There is little difference in the radial positions of the vortices other than the offset in root vortex location caused by the approximations made regarding blade planform.

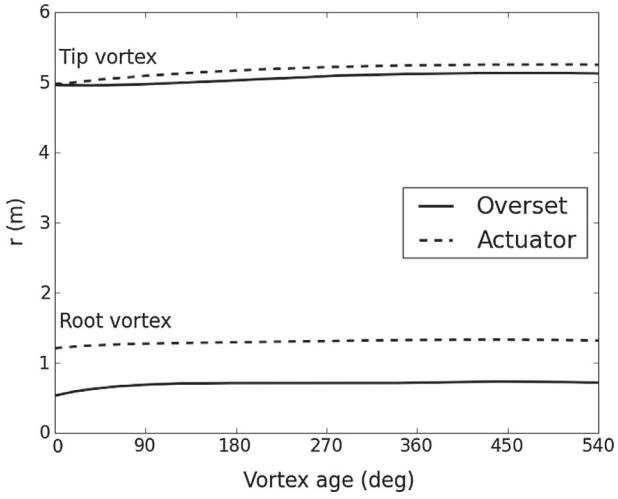
In addition to the wake features, it is also possible to compare the blade loading for the actuator blade model with the full overset simulation. Fig. 10 shows the variation of normal force coefficient along the span of the blade for the two cases. For the overset simulation, the normal force coefficient was extracted from the pressure distribution, whereas for the actuator blade simulation the normal force coefficient is a function of the sectional lift, drag, and local angle of attack according to Eqs. (1) and (2). The sectional normal force is defined as the component of sectional force which is normal to the rotor plane.

There are some differences between the normal force coefficients for the overset and actuator blade simulation, particularly near the tip region, where the tip loss effects are not captured well by the actuator blade model. In this approach, the focus is to develop and demonstrate a cost-effective actuator blade model of a rotating system. The simple a posteriori application of a tip loss correction used in blade element theory [40] correctly mimics the blade loading at the tip. Implementation of the tip correction during the simulation in a closed-loop fashion may have an influence on the inboard loads. Additional improvements are also possible by using a lookup table, possibly with 3-D corrections, instead of the unsophisticated loading model given by Eq. (1).

Further improvements in the accuracy of the actuator blade sectional loads can be realized by linking with available rotorcraft analysis tools. For example, comprehensive analysis codes such as DYMORE, RCAS, and CAMRADII which are already coupled with FUN3D and other rotorcraft CFD codes can be used for more accurate sectional load prediction in the blade element model



(a) Axial position, z of vortices vs. vortex age.



(b) Radial position, r of vortices vs. vortex age.

Fig. 9. Trajectories of root and tip vortices for both overset and actuator blades. The radial separation between the overset and actuator root vortices is approximately equal to the distance between the overset and actuator blade roots.

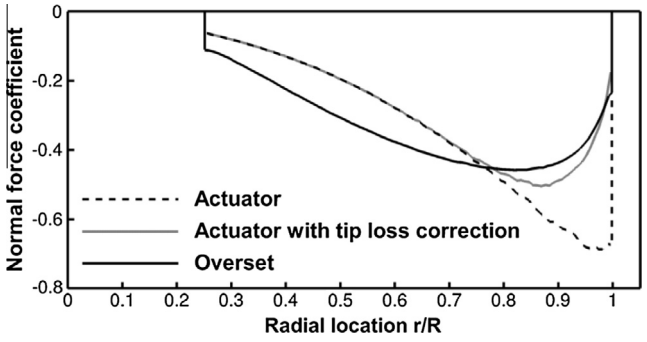


Fig. 10. Comparison of normal force coefficient for overset and actuator blades.

[18,19,31]. Comprehensive codes also facilitate aeroelastic simulation of rotating systems. Aeroelastic effects are important for structural analysis and also because they influence the aerodynamic loading of the blades. In the present work, a simple model of the sectional loads was used whereby loads are distributed uniformly along the chord. In the sophisticated approach discussed in prior efforts (i.e., Refs. [18,19,31]), the loading could be distributed non-uniformly to model the suction peak and match the sectional moment coefficient for the airfoil.

The computational cost of the overset and actuator simulations can be compared via CPU time per degree of freedom per time step. That is:

$$\text{cost} = \frac{(\text{wall time})(\text{no. CPUs})}{(\text{no. mesh nodes})(\text{no. time steps})} \tag{6}$$

The overset simulation required a total wall time of 13.1 h on 320 CPUs, and the actuator simulation required a wall time of 8.9 h on 96 CPUs, or 20% of the total cost in CPU hours. Both were computed for 1440 time steps (2 revolutions). Using the above definition, the actuator blade simulations have a cost of 8.7×10^{-4} CPU-seconds/DOF/step, while the overset simulations have a cost of 1.3×10^{-3} CPU-seconds/DOF/step. Note that this measurement is specific to the particular computer used for the simulations. The majority of the difference in cost is due to the difference in the degrees of freedom required to adequately resolve the flowfield. The basic CFD solution algorithm (calculation of residuals and Jacobian, solution of a linear system, etc.) is the same for both simulations. In addition to requiring extra degrees of freedom to resolve blade surfaces, the overset simulation incurs the cost of overset communication and mesh movement, which is typically on the order of 15% of the total computational expense.

The actuator blade method was originally developed for predicting unsteady loads on helicopter fuselages [18,19]. An analogous problem for wind turbines is predicting loads on a turbine tower and nacelle, but those loads are of little engineering interest compared with the inertial loads of the spinning blades. However, since the actuator blade method captures the gross features of the turbine wake, it shows potential for use in predicting interactions between multiple turbines in a wind farm. An upstream rotor with actuator blades (or its steady-state counterpart, the actuator disc) can be applied to predict the power loss experienced by a downstream turbine in its wake. Actuator blades could also approximate an unsteady upstream wake feeding into a high fidelity overset simulation downstream. In that manner, blade vortex interaction between a turbine and its upstream neighbors could be predicted. The principal difficulty in such a simulation would be the computational expense of running the simulation long enough to allow the wake to convect from one turbine to the next and the requirement of fine enough grid resolution between them to avoid dissipating the wake.

In order to address the issue of wake dissipation in the region downstream of the turbine, an adaptive grid refinement algorithm was implemented. The adaptive grid approach is similar to the one used successfully for rotating-system simulations in the Helios solver [41,42], but it operates on tetrahedral meshes and works on both near-body and background grids. This capability is present in FUN3D for steady and unsteady simulations on single or overset grids using a metric-based adaptation approach. The goal of the metric-based grid adaptation approach is to refine the grid selectively in regions where it is needed, and also to remove grid vertices where possible, based on the features of the solution computed on a coarser baseline grid. Using the computed solution, the grid is refined in an efficient manner, adding vertices only where they are required in the flow field. The metric may be based on vorticity magnitude, Q, pressure, or other quantities. For unsteady simulations, the metric is built up over a fixed time interval before adaptation occurs. Further information on metric-based adaptation in FUN3D is given by Park and Darmofal [43] and Shenoy [44].

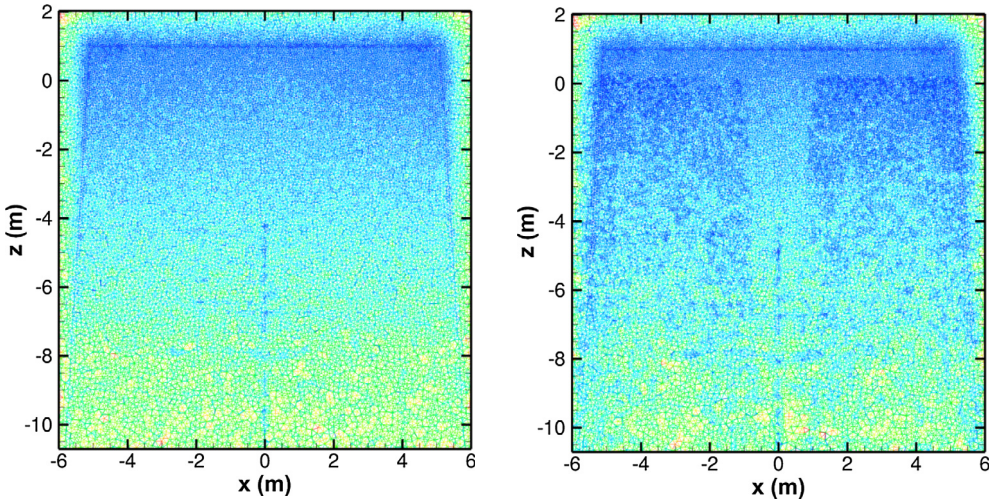
In the present study, the time-dependent adaptation metric is constructed using the vorticity-magnitude indicator, according to

the guidelines given by Shenoy et al. [45]. To generate the metric, several revolutions of the actuator blade model are simulated on the baseline grid to ensure that the wake is periodic, and then the metric is computed on the grid during one half of a revolution for the two-bladed turbine rotor. The adapted grid is then created using information from the metric. In this study, this adaptation procedure is performed twice; after the first adapted grid is created, the simulation is re-run using this adapted grid, and a new adaptation metric is constructed during the last one-half revolution. Finally, the simulation is re-run yet again using the new adapted grid. The goal of an iterative procedure like this one is to converge on a grid that is able to resolve critical features of the solution to a desired level while adding vertices only where they are needed. The obvious downside of the iterative procedure is that it is expensive; however, in general it is less expensive in terms of both engineering hours and computational hours than performing a full grid convergence study.

Visualizations of the three grids used in this adaptation study are given in Fig. 11. In this figure, the grids are sliced along the x–z plane after the full four-revolution simulation has been performed, and cells are colored by cell area in the sliced plane. The baseline grid has 2.4 million vertices; after the first adaptation the number increases to 3.5 million, and after the second adaptation the final adapted grid has 5.5 million vertices. Each level of adaptation demonstrates that vertices are being added primarily in the regions of the wake where vorticity magnitude is large. Directly below the rotor, a significant number of vertices are added across the span. Farther downstream, most of the added points are located near the root and tip region because the shed vorticity has rolled up into the traditional vortex structure (Fig. 8). In both adapted grids, significant refinement can be seen greater than two rotor radii downstream.

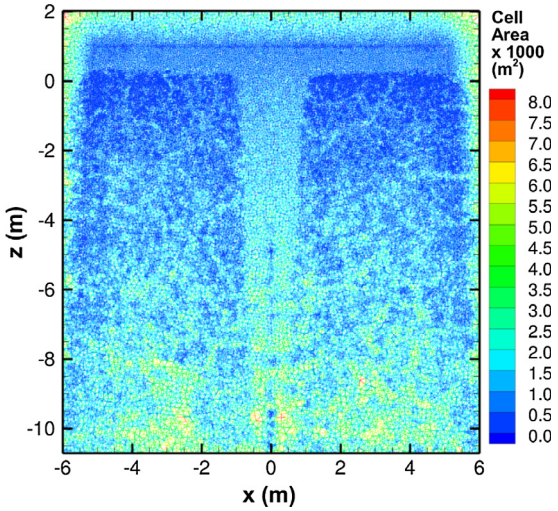
Fig. 12 shows the trajectories of the root and tip vortices for the simulations using the baseline and adapted grids. There is little noticeable difference in vortex trajectories between the three grids. This result suggests the baseline grid is already sufficiently refined to accurately predict the trajectories. If the vortex trajectories were the quantities of interest in the simulation, the first adaptation would have shown the baseline grid to be fine enough, and the second adaptation would not have been needed.

A more telling indicator of the quality of the wake prediction is the preservation of vortex strength as a function of vortex age. This



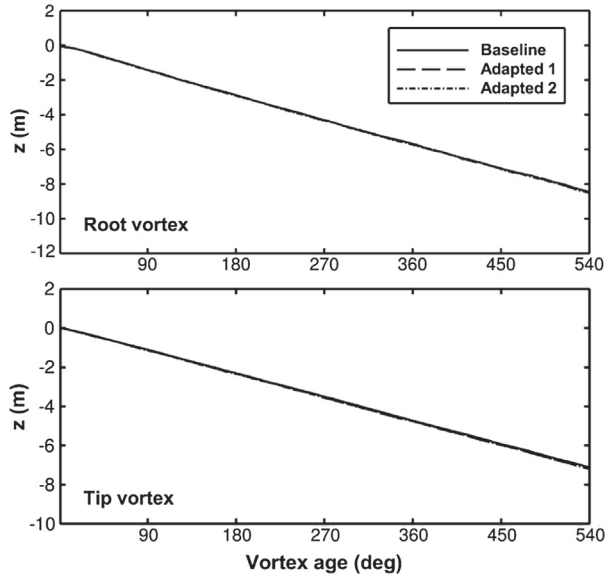
(a) Baseline grid, with 2.4 million vertices

(b) Adapted grid 1, with 3.5 million vertices

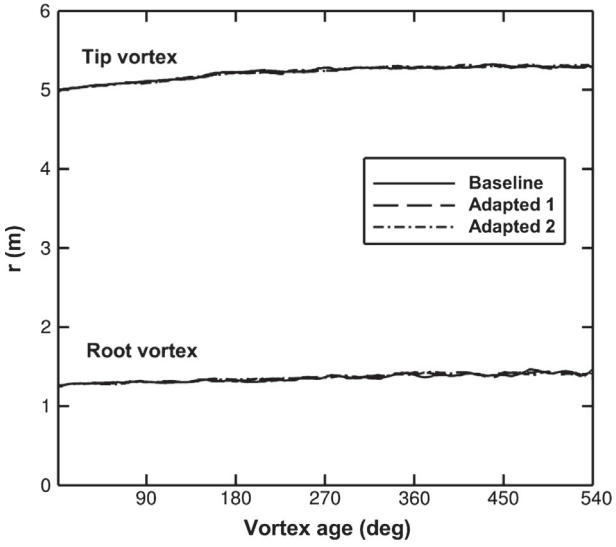


(c) Adapted grid 2, with 5.5 million vertices

Fig. 11. x–z slices of the baseline and adapted grids, with cell edges colored by contours of cell area. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Axial position, z of vortices vs. vortex age.



(b) Radial position, r of vortices vs. vortex age.

Fig. 12. Trajectories of root and tip vortices for the baseline and adapted grids.

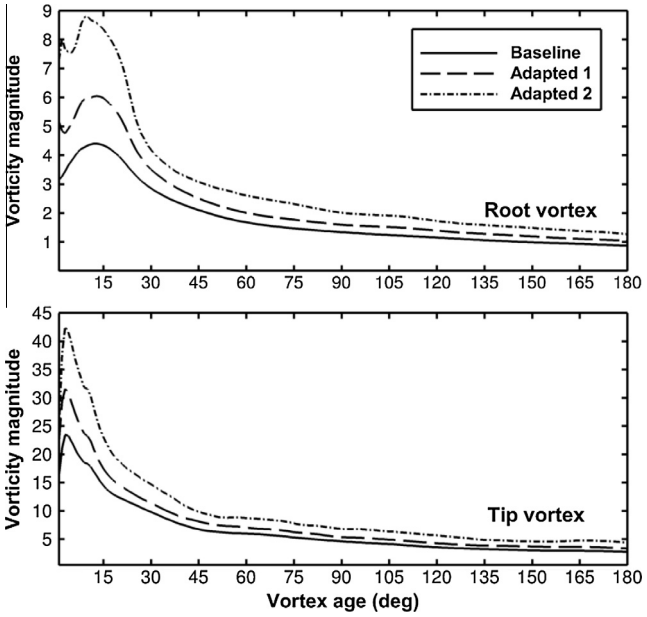
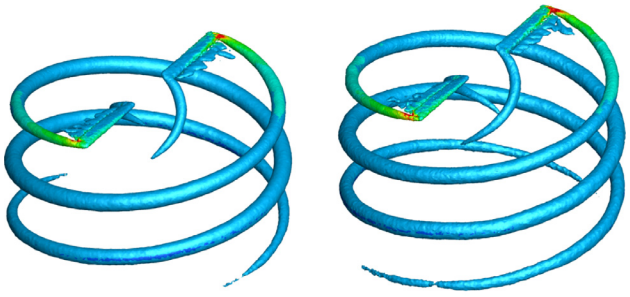
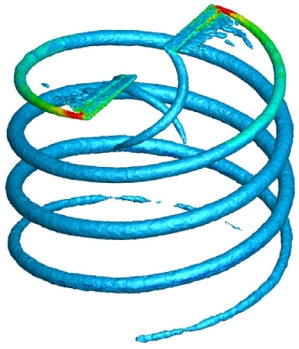


Fig. 13. Vorticity magnitude, normalized by R and U_{tip}, at the core of the root and tip vortices for the baseline and adapted grids.



(a) Baseline grid (b) Adapted grid 1



(c) Adapted grid 2

Fig. 14. $Q = 0.05$ iso-surfaces normalized by R and U_{tip} at $V_\infty = 7$ m/s. All cases were run for four revolutions at 1° azimuth per step.

comparison is shown in Fig. 13, which tracks the magnitude of vorticity at the core of the root and tip vortices over the first 180 degrees of vortex age. For both vortices, the vortex core is stronger in the adapted grids throughout the simulation. Notably, the greatest difference occurs within the first thirty degrees of vortex age while vorticity along the span of each actuator blade rolls up to form the root and tip vortices. This rollup occurs more quickly for the tip vortex than the root vortex. Since there are still significant differences in the root vortex strength for low wake age, further iterations of grid adaptation would be required for this feature to be fully resolved.

Iso-surfaces of Q can also help to quantify wake preservation. In Fig. 14, $Q = 0.05$ iso-surfaces are shown for the baseline and two adapted grid simulations. All simulations were run for four revolutions at one degree azimuthal advance per time step, and the Newton subiterations converged to the same level. The first adapted grid shows a modest improvement in wake age that is visible at this level of Q (roughly 45 degrees for the tip vortex). The second adapted grid shows a significant improvement, with the tip vortex visible for greater than 180 additional degrees compared to the baseline grid. This result indicates a significantly lower level of wake dissipation on the adapted grids.

6. Conclusions

In this work, a practical new method to rapidly compute the flow field around a complex rotating system is demonstrated.

Simulation of wind turbine blades using a discrete source representation removes the need to include the actual rotating blade surfaces and associated near-body grids. Thus, the overall grid size is reduced and the requirement of overset moving grids in the simulation is eliminated. An improved nearest neighbor search algorithm based on kd-trees is presented here, which is shown to dramatically improve the speed of source-to-node association with a minimal impact on memory usage. This new search algorithm allows each blade to be modeled individually as a moving cloud of sources while maintaining computational efficiency sufficient for practical use. Actuator blade simulations of the NREL Phase VI Unsteady Aerodynamics Experiment at 7 m/s were compared against high-fidelity overset simulations that model the real blade geometry to evaluate the approach. Key findings of this study include:

- Tip and root vortex trajectories in the near wake predicted by the actuator blade method are in very close agreement with predictions of the full overset method. The tip vortex trajectory is nearly identical between the two methods, but there are some differences in the root vortex position due to geometric differences at the blade root.
- The actuator blade simulation required only 20% the computational cost, in terms of total CPU hours, of the overset simulation. The dramatic cost reduction is due mostly to the fact that fewer total grid nodes required for the actuator blade simulation. Additional savings are also afforded because the need to move a large overset grid is not needed in the actuator blade method.
- Blade loading predictions for the actuator blade method can be improved by incorporating a tip loss model and airfoil lookup tables.
- Grid adaptation can be used with the actuator blade method to preserve wake features for long distances downstream while making efficient use of added grid vertices. Possible applications of this grid adaptation work include turbine-to-turbine interaction studies.

With minimal modifications, the actuator blade algorithm could be extended to include the effects of blade flexibility or active control surfaces. This would be a generalization of the rigid flapping motion of actuator sources already available in the model. The actuator blade method would provide an additional significant computational benefit for aeroelastic analyses because it eliminates the need to deform a large overset blade grid. Because the wind turbine blades are slender, high aspect ratio components, they can be modeled as flexible beams, which permits the use of CFD/CSD coupling with comprehensive or multi-body dynamics codes that have already been designed to output blade motion for fully overset simulations. This process is routinely applied in rotorcraft applications [33,38,46]. Coupling with a comprehensive code could also provide finite state unsteady aerodynamic predictions (e.g. via the finite state theory of Peters et al. [47]) that are substantially more accurate than the simple linear blade element model that is currently used to determine source strengths.

Acknowledgements

This work was supported in part by the National Science Foundation, Project 0731034, "Advancing Wind Turbine Analysis and Design for Sustainable Energy". The authors would like to thank NSF and the NSF Program Officers, Trung Van Nguyen and Greg Rorrer, for their support in this endeavor. Computational support for this research was supported in part by the National Science Foundation through TeraGrid [48] (Grant # TG-MSS080017N). TeraGrid resources were hosted by NCSA (Mercury and Abe). The authors would like to thank Christopher Stone for his assistance in with obtaining Teragrid time, Scott Schreck from NREL for providing and explaining the experimental datasets, and Ross Cooper for providing the grids and initial simulations for the grid adaptation study.

References

[1] Johansen J, Madsen HA, Sørensen NN, Bak C. Numerical investigation of a wind turbine rotor with an aerodynamically redesigned hub region. In: 2006 European wind energy conference and exhibition, Athens, Greece; 2006.
[2] Zahle F, Sørensen NN. Overset flow simulation on a modern wind turbine. In: 26th AIAA applied aerodynamics conference, AIAA-2008-6727: Honolulu, HI; 2008.
[3] Potsdam M, Mavriplis D. Unstructured mesh CFD aerodynamic analysis of the NREL phase VI Rotor. In: 47th AIAA aerospace sciences meeting, AIAA-2009-1221: Orlando, FL; 2009.
[4] Hansen MOL et al. State of the art in wind turbine aerodynamics and aeroelasticity. Prog Aerospace Sci 2006;42:285–330.
[5] Heege A, Betran J, Radovcic Y. Fatigue load computation of wind turbine gearboxes by coupled finite element, multi-body system and aerodynamic analysis. Wind Energy 2007;10:395–413.
[6] Chattot J. Helicoidal vortex model for wind turbine aeroelastic simulation. Comput Struct 2007;85:1072–9.
[7] Leishman JG. Challenges in modeling the unsteady aerodynamics of wind turbines. In: 21st ASME wind energy symposium and 40th AIAA aerospace sciences meeting and exhibit, AIAA-2002-0037: Reno, NV; 2002.
[8] Schreck S. Low frequency shedding prompted by three dimensionality under rotational augmentation. In: 48th AIAA aerospace sciences meeting and exhibit, AIAA-2010-0640: Orlando, FL; 2010.
[9] Tangler, JL. The nebulous art of using wind-tunnel airfoil data for predicting rotor performance. Technical report NREL/CP-500-31243, National Renewable Energy Laboratory, Golden, CO; January 2002.
[10] Smith M, Liggett N, Koukol BC. The aerodynamics of airfoils at high and reverse angles of attack. AIAA J Aircraft 2011;48(6):2012–23. http://dx.doi.org/10.2514/1.55358.
[11] Huyer SA, Simms D, Robinson MC. Unsteady aerodynamics associated with a horizontal-axis wind turbine. AIAA J 1996;34(7):1410–9.
[12] Coton FN, Wang T, Galbraith RAM. An examination of key aerodynamic modeling issues raised by the NREL blind comparison. In: 21st ASME wind energy symposium and 40th AIAA aerospace sciences meeting and exhibit, AIAA-2002-0038: Reno, NV; 2002.
[13] Fejtek I, Roberts L. Navier–Stokes computation of a wing/rotor interaction for a tilt rotor in hover. AIAA J 1992;30(11):2595–603.
[14] Sørensen JN, Myken A. Unsteady actuator disc model for horizontal axis wind turbines. J Wind Eng Indust Aerodyn 1992;39:139–49.
[15] Sørensen JN, Kock CW. A model for unsteady rotor aerodynamics. J Wind Eng Indust Aerodyn 1995;58:259–75.
[16] Chaffin MS, Berry JD. Helicopter fuselage aerodynamics under a rotor by Navier–Stokes simulation. J Am Helicopter Soc 1997;42(3):235–42.
[17] Le Chuiton F. Actuator disc modeling for helicopter rotors. Aerospace Sci Technol 2004;8:285–97.
[18] O'Brien DM, Smith MJ. Analysis of rotor-fuselage interactions using various rotor models. In: AIAA 43rd aerospace sciences meeting, AIAA-2005-468, American Institute of Aeronautics and Astronautics: Reno, NV; 2005.
[19] O'Brien DM. Analysis of computational modeling techniques for complete rotorcraft configurations. PhD thesis, Georgia Inst. of Technology; 2006.
[20] Sørensen JN, Shen WZ. Numerical modeling of wind turbine wakes. J Fluids Eng 2002;124:393–9.
[21] Shen WZ, Zhang JH, Sørensen JN. The actuator surface model: a new Navier–Stokes based model for rotor computations. J Sol Energy Eng 2009;131.
[22] Mikkelsen R. Actuator disc methods applied to wind turbines. PhD thesis, Technical University of Denmark; 2003.
[23] Michelsen JA. Basis3D – a platform for development of multiblock PDE solvers. Technical report AFM 92-05, Technical University of Denmark, DTU; 1992.
[24] Sørensen NN. General purpose flow solver applied to flow over hills. PhD thesis, Risø National Laboratory, Roskilde, Denmark; 1995.
[25] Ivanell S, Sørensen JN, Mikkelsen R, Henningson D. Analysis of numerically generated wake structures. Wind Energy 2009;12:63–80.
[26] Bonhaus D. An upwind multigrid method for solving viscous flows on unstructured triangular meshes. Master's thesis, George Washington University; 1993.
[27] Anderson W, Rausch R, Bonhaus D. Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids. J Comput Phys 1996;128(2):391–408.
[28] Chorin A. A numerical method for solving incompressible viscous flow problems. J Comput Phys 1967;2(1):12–26.
[29] Noack R. DiRTlib: a library to add an overset capability to your flow solver. In: 17th Computational fluid dynamics conference, AIAA-2005-5116, American Institute of Aeronautics and Astronautics: Toronto, Ontario; 2005.
[30] Noack R. SUGGAR: a general capability for moving body overset grid assembly. In: 17th Computational fluid dynamics conference, AIAA-2005-5117, American Institute of Aeronautics and Astronautics: Toronto, Ontario; 2005.

[31] Renaud T, O'Brien DM, Smith MJ, Potsdam M. Evaluation of isolated fuselage and rotor-fuselage interaction using CFD. J Am Helicopter Soc 2008;53(1):3–17.
[32] Biedron R, Vatsa V, Atkins H. Simulation of unsteady flows using an unstructured Navier–Stokes solver on moving and stationary grids. In: Proceedings of the 23rd AIAA applied aerodynamics conference, AIAA-2005-5093, Toronto, Ontario, Canada; 2005.
[33] Lynch CE, Smith MJ. Unstructured overset incompressible computational fluid dynamics for unsteady wind turbine simulations. Wind Energy 2013;16(7). http://dx.doi.org/10.1002/we.1532.
[34] O'Brien DM, Smith MJ. Understanding the physical implications of approximate rotor methods using an unstructured CFD method. In: 31st European rotorcraft forum, Florence, Italy; 2005.
[35] Bentley JL. Multidimensional binary search trees used for associate searching. Commun ACM 1975;18:509–17.
[36] Friedman JH, Bentley JL, Finkeyl RA. An algorithm for finding best matches in logarithmic expected time. Technical report STAN-CS-75-482, Stanford CS Rep.; February 1975.
[37] Hand MM, Simms DA, Fingersh LJ, Jager DW, Cotrell JR. Unsteady aerodynamics experiment phase VI: wind tunnel test configurations and available data campaigns. Technical report NREL/TP-500-29955, National Renewable Energy Laboratory, Golden, CO; December 2001.
[38] Abras JN, Lynch CE, Smith MJ. Computational fluid dynamics-computational structural dynamics rotor coupling using an unstructured reynolds-averaged Navier–Stokes methodology. J Am Helicopter Soc 2012;57(1):1–14.
[39] Vatsa V, Carpenter M. Higher order temporal schemes with error controllers for unsteady Navier–Stokes equations. In: 17th AIAA computational fluid dynamics conference, AIAA-2005-5245, Toronto, Ontario, Canada; 2005.
[40] Leishman JG. Principles of helicopter aerodynamics. 2nd ed. Cambridge: Cambridge University Press; 2006.
[41] Wissink A, Sitaraman J, Sankaran V, Mavriplis D, Pulliam TH. Multi-code python-based infrastructure for overset CFD with adaptive cartesian grids. In: 46th AIAA aerospace sciences meeting and exhibit, AIAA-2008-0927: Reno, NV; 2008.
[42] Sitaraman J, Katz A, Jayaraman B, Wissink A, Sankaran V. Evaluation of a multi-solver paradigm for CFD using unstructured and adaptive cartesian grids. In: 46th AIAA aerospace sciences meeting and exhibit, AIAA-2008-0660: Reno, NV; 2008.
[43] Park M, Darmofal D. Parallel anisotropic tetrahedral adaptation. In: AIAA 46th aerospace sciences meeting and exhibit, AIAA-2008-0917, Reno, NV; 2008.
[44] Shenoy R, Smith M, Park M. Unstructured overset mesh adaptation with turbulence modeling for unsteady aerodynamic interactions. J Aircraft, 2013.
[45] Shenoy R, Holmes M, Smith M, Komerath N. Scaling evaluations on the drag of hub system. J Am Helicopter Soc 2013;58(3):1–13.
[46] Biedron RT, Lee-Rausch EM. Rotor airloads prediction using unstructured meshes and loose CFD/CSD coupling. In: 26th AIAA applied aerodynamics conference, AIAA-2008-7341, Honolulu, HI; 2008.
[47] Peters DA, Karunamoorthy S, Cao WM. Finite state induced flow models part I: two-dimensional thin airfoil. J Aircraft 1995;32(2):313–21.
[48] Catlett C. *TeraGrid*: analysis of organization, system architecture, and middleware enabling new types of applications. In: Grandinetti Lucio, editor. HPC and grids in action. Amsterdam: IOS Press 'Advances in Parallel Computing' series; 2007.