Twelfth International Conference on
Computational Fluid Dynamics (ICCFD12),
Kobe, Japan, July 14-19, 2024

A Wall-Distance Method for Turbulence Modeling

X. Zhang *, B. Diskin**, A. Walden**, and E. J. Nielsen**
Corresponding author: xinyu.zhang@nasa.gov
* Analytical Mechanics Associates, USA.
** NASA Langley Research Center, USA.

Abstract: The distance from a grid point to the closest wall surface, wall distance, is a fundamental quantity in turbulence modeling. Efficiency of wall-distance calculations has become more critical as the size of computational grids has significantly increased in recent years. This paper reports on an initial implementation of a new search-based wall-distance method that is suitable for general unstructured computational fluid dynamics (CFD) grids and tailored for requirements specific for turbulence modeling. The method uses a two-step approach to calculate the wall distance. In the first step, the wall distance is approximated for each grid point as the minimum distance from this point to a vertex of a triangular face at the wall. The point-to-vertex distance calculation is relatively inexpensive but may lead to a significant error in the wall-distance approximation, especially for grid points near the wall. In the second step, for grid points located within a predefined distance (*threshold*) from the wall, the wall distance is computed as the minimum distance to wall faces. As a result, the wall distance is exact for all grid points within the threshold. This two-step approach reduces the computational cost, yet achieves high and controllable accuracy in the evaluation of the wall distance. Algorithmic enhancements are presented to improve the efficiency of wall-distance computations. A comprehensive assessment of the new method is reported for large-scale unstructured CFD grids generated for the Fifth AIAA CFD High-Lift Prediction Workshop. The performance of the new wall-distance method compares favorably with the performance of two established methods implemented in high-performance CFD codes.

Keywords: Wall Distance, High Performance Computing, Computational Fluid Dynamics, Turbulence Modeling.

1 Introduction

The wall distance is the minimum distance from a point to a wall surface, and it is a fundamental quantity in turbulence modeling. In computational fluid dynamics (CFD), unstructured grids are often used for the simulation of turbulent flows in the presence of complex geometries. For such simulations, the wall is typically tessellated with triangular elements (faces). The wall distance is required for points of a three-dimensional (3D) unstructured grid generated on a computational domain of interest. Efficiency of wall-distance calculations has become more critical as the size of computational grids used for simulating turbulent flows has significantly increased in recent years.

The topic of computing the minimum distance from a query point to a surface has been studied extensively in the literature for applications in many areas [1, 2, 3, 4]. The most efficient algorithms are tuned to application-specific assumptions and requirements. For example, in computer graphics, it is often assumed that accuracy at all query points is equally important, that the number of query points is small in comparison to the number of surface faces, and that surface faces are isotropic triangles whose size is determined by the surface curvature. Scalability is achieved by careful balancing of the surface data between different partitions.

Wall-distance computations for turbulent flows also have specific features and requirements. In a typical CFD grid, there are many more query points than wall faces. In turbulence-model equations, the wall distance often appears in the denominator of a source term. Therefore, grid points located near the wall require much higher wall-distance accuracy than grid points that are far from the wall. In distinction

from computer-graphics applications, the wall tessellation for CFD applications can be highly anisotropic and is not exclusively governed by curvature and geometry features. The size, shape, and distribution of wall faces reflects expectations for accurate flow representation and may not correlate with the wall curvature. For example, in anticipation of shocks, multiphase flows, or contact discontinuities, small, highly anisotropic faces may appear even on flat walls.

The wall distance is an auxiliary computation for simulating turbulent flows and is expected to take a small fraction of the simulation time. However, efficient wall-distance computations in a highly parallel environment are challenging. Typically, all grid points must have the wall distance calculated before starting flow simulations. For flow simulations on a stationary grid, it is a one-time computation usually conducted at the preprocessing stage. For unsteady flows on dynamic, deforming grids, the wall distance may need to be updated at each time step. For scalable computations, most modern CFD solvers rely on a domain-decomposition approach for coarse-grained parallelism where a Message Passing Interface (MPI) implementation of the Message Passing Standard is used for data communication. In this paradigm, each grid partition is assigned to an MPI rank. In typical CFD computations, grid partitioning is done to optimize performance of the flow solver. There is no expectation of the load being balanced for wall-distance computations. For parallel efficiency, memory management is also critical. In a CFD grid, one can expect hundreds of millions of grid points and millions of wall faces. CFD solvers that have been developed for stationary-grid applications either store a copy of the wall grid on each rank or load portions of the wall grid on each rank one at a time. These approaches are not scalable and not suitable for modern computer architectures using Graphics-Processing-Unit (GPU) accelerators.

In recognition of the wall-distance challenges encountered in modern CFD simulations, there is a renewed interest in developing wall-distance methods suitable for high-performance CFD computing. Reference [5] gives a concise overview of popular approaches used in CFD codes today and suggests its own method. The approaches are classified into three groups:

- Search algorithms [6, 7, 8, 9]
- Solutions of partial differential equation (PDE) [10, 11, 12]
- Domain painting [13, 14, 15]

A search algorithm typically considers one query point at a time and uses a nearest-neighbor search to find the distance to the wall. An exhaustive nearest-neighbor search, i.e., a search that, for each query point, computes the distances to all wall faces, is prohibitively expensive. A more efficient nearest-neighbor algorithm usually constructs a bounding volume hierarchy (BVH) [16, 17] of wall faces. BVH is a tree structure in which each leaf node is a wall face wrapped in a bounding volume. The faces are then recursively grouped in progressively expanding sets that are enclosed within bigger bounding volumes. Eventually, the entire tree structure is enclosed within a single bounding volume. Once a BVH is constructed, it greatly expedites the search by quickly eliminating portions of the wall that do not help determining the minimum distance for a given query point. Various types of bounding volumes have been proposed and tested. The choice of bounding volume is a trade-off between simplicity and fitting tightness. On one hand, a simple bounding volume enables a fast distance computation from a query point to the bounding volume. On the other hand, tight bounding volumes are expected to eliminate larger portions of the wall. Commonly used bounding volumes include axis-aligned bounding boxes [18, 19], bounding spheres [20, 21, 22], oriented bounding boxes [23], and discrete orientation polytopes [24]. A BVH search is efficient for identifying the distance from a query point to a set of discrete points, e.g., vertices of wall faces. However, this distance may not be the exact wall distance. The latter requires computing distance to wall faces. Computing the point-to-face distance is significantly more expensive than computing the point-to-vertex distance. With BVH, point-to-face wall-distance computations are still relatively efficient for query points that are near the wall. For points that are far from the wall, there may be many wall faces at a similar distance from the query point, resulting in smaller portions of the wall eliminated from the BVH search. A compute rank containing many query points that are far from the wall may require significantly more computations than a rank that has most of its query points near the wall.

An alternative is a PDE-based approach. The wall distance can be considered as the time taken by a unit-speed wave front emanating from the wall to reach a given query point. A PDE describing wave propagation can be used for computing the wall distance. Such a PDE can be discretized and solved using standard methods. PDE solvers are scalable; however, the accuracy of PDE solutions is limited by the discretization error and iterative convergence. The exact wall distance can be achieved only in the limit of grid refinement and zero residuals. Thus, the wall-distance computation may require

generation of auxiliary grids. The accuracy and iterative convergence of PDE solutions deteriorate near sharp geometrical features where wall-distance accuracy is most important for turbulence modeling.

Painting methods "color" parts of the computational domain that are closest to particular primitive surface elements (faces, edges, vertices) of the wall tessellation. The painting can be topological, based on grid connectivity, or grid independent, based on Voronoi diagrams. Topological painting is typically implemented as an advancing front and can be perceived as a topological equivalent of the wave-propagation approach. In the presence of non-uniform grids and highly varying cell sizes, the wall distance can accumulate errors. Because of their sequential propagation nature, advancing-front methods are generally not scalable. Grid-independent painting uses Voronoi diagrams. Although Voronoi diagrams precisely encode the proximity to a primitive surface element, the construction/search of Voronoi diagrams is computationally intensive.

The method proposed in Ref. [5] combines characteristics of search-based and PDE-based approaches. It computes the wall distance for individual grid points, uses face bounding boxes (voxelized surface), and simulates wave propagation (expanding voxelized spheres). The wave propagation is performed on an auxiliary Cartesian grid. Generation of an additional grid for computing the wall distance appears to be a serious complication for a wall-distance algorithm intended for applications on general unstructured grids.

In summary, search-based approaches can provide the true value of the wall distance, but may be inefficient in treating grid points distant from the wall. PDE-based approaches are scalable, but may require generation of auxiliary grids and may lose accuracy in places that are important for turbulence modeling. Painting methods are not scalable and may not be sufficiently accurate or efficient.

The method proposed in this paper is a search algorithm that targets simulations of turbulent flows on unstructured grids and is tailored for specific turbulence-modeling requirements. In particular, the method uses the same partitions of the flow solver, limits searches on each rank to the wall data (vertices and faces) that have a chance to be the closest ones for a grid point on the rank, and applies different wall-distance computations for points that are near and far from the wall. A version of the method has been implemented in FUN3D [25], a large-scale, unstructured-grid CFD solver developed at NASA Langley Research Center (LaRC). The implementation of the new wall-distance method is a work in progress. In the current version, identification of the wall data that needs to be searched on each rank occurs after the entire wall data have been gathered at the rank. In the final version, only the required wall data will be gathered on each rank. Even in the current version, performance of the new method has been compared with the performance of established wall-distance methods [26, 27] and found to be superior when applied to unstructured, mixed-element, highly anisotropic grids generated for solving the Reynolds-averaged Navier-Stokes (RANS) equations for the Fifth AIAA CFD High-Lift Prediction Workshop (HLPW5) [28].

The material in this paper is organized as follows. Section 2 describes the configurations, flows, and grids used in this paper for the illustration and assessment of wall-distance methods. A brief description of the reference wall-distance methods is provided in Section 3. Section 4 presents details of the new method. Section 5 reports on numerical tests assessing the performance and accuracy of the new method. A summary and future plans are presented in Section 6.

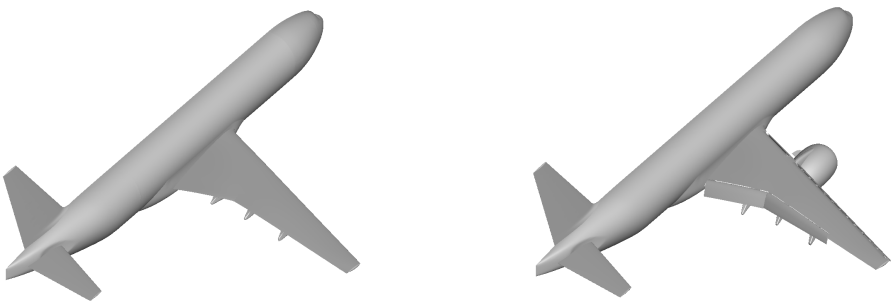
2 Configurations, Flow Conditions, and Grids

The High-Lift Common Research Model (CRM-HL) [29] is an open-source, publicly-available commercial transport aircraft geometry in a high-lift configuration that is used for CFD validation within a broad international community and studied in HLPW5 [28]. The reference characteristics of the full-scale model are shown in Table 1. Specifically, configurations 2.1 and 2.4 (see Fig. 1) are considered in this paper. Configuration 2.1 is a wing-body-tail configuration with horizontal and vertical stabilizers but without high-lift devices. Configuration 2.4 is the same configuration with the addition of slats, flaps, and a nacelle/pylon.

The method proposed in the current paper provides the exact wall distance for all points that are close to the wall. The user can specify the distance (*threshold*) from the wall, within which the wall distance is computed to wall faces. It is expected that the domain bounded by *threshold* includes the attached boundary layer. The thickness of the boundary layer is a function of the flow Reynolds number, the extent of the geometry, and any separation-related concerns. Flows considered for configurations 2.1 and 2.4 of HLPW5 have Reynolds numbers based on the mean aerodynamic chord (MAC), $Re_{MAC} = 5.4 \times 10^6$ and $Re_{MAC} = 5.9 \times 10^6$, respectively. For an attached flat-plate turbulent boundary layer, the thickness, τ,

Table 1: Reference quantities of CRM-HL

Mean aerodynamic chord	275.8 inches
Semi-span model reference area	297,360.0 in^2
Fuselage extent	2533.43205 inches
x moment center	1325.9 inches
y moment center	0.0 inches
z moment center	177.95 inches



(a) Configuration 2.1. (b) Configuration 2.4.

Figure 1: HLPW5 configurations.

is estimated [30] as

$$\tau \approx 0.37 \frac{L}{Re_L^{\frac{1}{5}}}, \tag{1}$$

where L is the characteristic length. Given the Reynolds numbers and the reference parameters listed in Table 1, the estimates of the boundary-layer thickness at the aft of the fuselage are 27.1 inches and 26.6 inches for configurations 2.1 and 2.4, respectively. These are rough estimates that assume a zero-pressure-gradient, self-similar, turbulent boundary layer. In most wall-distance computations presented in this paper, *threshold* is twice the Eq. 1 estimate, i.e., $threshold = 54.2$ inches is used for configuration 2.1, and $threshold = 53.2$ inches is used for configuration 2.4.

The unstructured, mixed-element grids used in this paper have been generated for HLPW5 using the Heldenmesh grid-generation tool developed and supported by the Helden Aerospace Corporation [31]. The wall surface is tessellated into triangular faces. The grid unit is inches. Tables 2 and 3 provide statistics for grid families generated for configurations 2.1 and 2.4, respectively. Grid names follow the HLPW5 website convention [28].

Table 2: Statistics of grids generated for configuration 2.1

Grid	Grid points (millions)	Wall vertices (millions)	Wall faces (millions)
C	1.482	0.062	0.123
M	3.869	0.127	0.253
F	24.051	0.468	0.936
G	53.848	0.816	1.630

Table 3: Statistics of grids generated for configuration 2.4

Grid	Grid points (millions)	Wall vertices (millions)	Wall faces (millions)
C	6.015	0.270	0.540
M	14.727	0.538	1.075
F	81.550	1.852	3.703
G	178.386	3.187	6.374

3 Reference Wall-Distance Methods

This section provides an overview of two established wall-distance methods that serve as references to assess the performance and accuracy of the new method. Both methods use a BVH based on a binary tree. The *legacy* method [26] in FUN3D has been used for a wide variety of applications. The second method is the wall-distance method implemented in NASA's mesh-adaptation tool, *refine* [27].

3.1 Legacy FUN3D method

The first reference method is described in Ref. [26]. This method has been extensively and successfully used in FUN3D RANS simulations for more than 20 years. The method calculates the wall distance for a grid point by locating the closest wall vertex and then examining the wall faces attached to the vertex. Let us consider the task of computing the wall distance for N_F grid points to a wall that contains N_S vertices. Axis-aligned bounding boxes are constructed to enclose the wall vertices. The method starts with a large bounding box containing all wall vertices and generates a binary tree of boxes by recursively bisecting each box in the direction of its longest side. The dividing plane is chosen to have half of the wall vertices lie on each side. This bisection stops (a box becomes a tree leaf) when the box has $\sqrt{N_S}$ or fewer wall vertices. For each grid point, the leaf boxes are sorted in the ascending order of the minimum distance to the grid point. The leaf boxes are visited in the order starting from the closest one. The distance from the grid point is computed to all wall vertices contained in the leaf box. The minimum point-to-vertex distance and the closest wall vertex are retained. If the minimum distance to the next leaf box is greater than the currently identified minimum point-to-vertex distance, the search stops because the closest wall vertex has been found. Once the closest wall vertex has been found, the distances from the grid point to the wall faces that share the closest wall vertex are calculated. The minimum point-to-face distance is set as the wall distance for the grid point. Assuming that the number of wall faces sharing a vertex is bounded (this assumption excludes grids with a polar singularity), the computational cost of this algorithm is $O(N_F\sqrt{N_S})$.

The legacy method is relatively efficient, but is known to miscalculate the actual minimal distance for some pathological cases. For example, Figure 2 shows the closest face, which is not attached to the closest wall vertex. In practice, the method works for a wide range of grids and geometries and has been extensively used in simulations of turbulent flows.

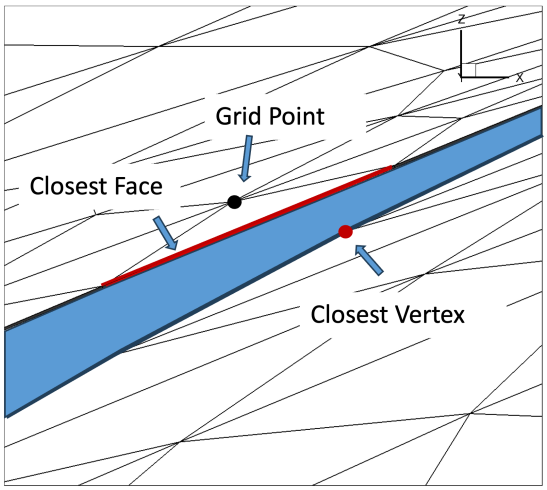


Figure 2: Minimum wall distance achieved at face not attached to closest vertex.

3.2 Wall Distance in *refine*

The second reference wall-distance method is the one implemented in the NASA mesh-adaptation tool *refine* [27]. This method computes the exact wall distance for all grid points and is used by FUN3D to compute the wall distance on adapted grids. The method uses a binary tree that defines a parent-child relation between wall faces [32]. Each node of the tree contains a face and its bounding sphere. The node also holds a radius of another concentric sphere that encloses faces of all its descendant nodes. In construction of the binary tree, faces are inserted in a predefined order. When a new face is inserted, the face starts from the root node and descends through the generations of tree nodes. When the face visits a node, the radius of the bounding sphere of the node is updated to include the face. If the node has less than two children, the face is assigned to a new child of the node. If the node has two children, then the face descends to the child node that is closer to the center of the face bounding sphere. After the binary tree has been completed, the wall distance is computed for each grid point. The point recursively traverses through the tree and measures the distance to the face bounding sphere. If the distance is less than the current wall distance, the distance from the point to the face is computed. The wall distance is updated as needed. Then the distance to its descendant bounding sphere is checked. If the distance is greater than the current wall distance, all its descendants are excluded from the search. Otherwise, the grid point proceeds to a next-generation node. At the end of this recursive tree traverse, the grid point has its exact wall distance. Note that the efficiency of this method depends on the ordering in which faces are introduced to the binary tree. A natural order based on spatial or topological proximity is not beneficial for the tree balance. In the *refine* wall-distance computations reported in this paper, a random permutation of faces is used. One can also use a self-balancing tree [33] to improve the tree balance.

4 New method

In a preprocessing stage, all the grid data including the wall data are loaded and partitioned between ranks. Each rank owns some grid points and possibly some wall faces. The number of grid points owned by different ranks is well balanced. The wall data is not balanced; some ranks may have many wall faces, while other ranks may have none. The method is expected to provide an estimate of the wall distance for all grid points. The main features of the new method are the following.

- This is a search-based method. Wall-distance computations are independently conducted for each grid point on a rank.
- Wall vertices and wall faces are searched separately. Each rank searches only wall vertices and wall faces that have a chance to be the closest one to a grid point on the rank.
- For all grid points, the initial wall-distance approximation is computed to wall vertices.
- For grid points within *threshold* from the wall, the wall distance is computed to wall faces.
- A nonzero wall-distance relative-error tolerance can be defined to speed up computations.

In the description of the new method, the following terminology is used. A *bounding box*, or *box* for brevity, is a an anisotropic axis-aligned cuboid that tightly encloses a subspace or a group of wall vertices or wall faces. Bounding boxes can overlap or leave uncovered gaps in the domain. Bounding boxes are usually organized in a tree structure. A *voxel* is also an anisotropic axis-aligned cuboid; the term is used to represent a unit volume in a regular partition of a 3D subdomain. A voxel partition covers the entire subdomain without overlap and can be organized as a 3D matrix, in which each individual voxel occupies a known space and can be referenced either by the i, j, k index or by a corresponding single-integer index. Each voxel also has an individual bounding box that may have a different size than the voxel itself and serves to represent tight bounds of vertices or faces registered with the voxel. Voxels of different partitions can create an octree with parent-child relations, in which a parent voxel of a "coarse" partition relates to the eight "fine"-partition child voxels that occupy the same space.

In the current implementation, the wall-distance method performs the following four major steps on each rank.

1. Collecting the global wall data.
2. Selecting the wall data necessary to compute the wall distance for grid points on the rank.
3. Computing the point-to-vertex wall distance for the grid points on the rank.

4. Computing the point-to-face wall distance for the grid points that are closer to the wall than *threshold*.

Details of the steps are described in Sections 4.1-4.4.

4.1 Collecting Global Data

Prior to this step, the grid points and the wall data are distributed between ranks. At this step, the wall data needed for computing the wall distance are gathered on each rank. In the current implementation, the entire global wall data are gathered on each rank. This approach is not scalable. In the final version of the new method, each rank will only collect the wall data necessary for computing the wall distance on the rank. The following steps are currently performed on each rank.

- The indices of the grid points located at the wall (wall vertices) are collected.
- The indices are sorted to a list containing unique vertices.
- The coordinates of these vertices are collected.
- The wall face indices are collected and linked to the indices of the wall vertices.
- If the amount of wall data is too large to upload on each rank, the wall data are divided into chunks. Chunks are gathered in turn. Each rank computes and updates the wall distance for each chunk. Some faces of a chunk may contain vertices from other chunks. Therefore, coordinates of face vertices are gathered after collection of each chunk.

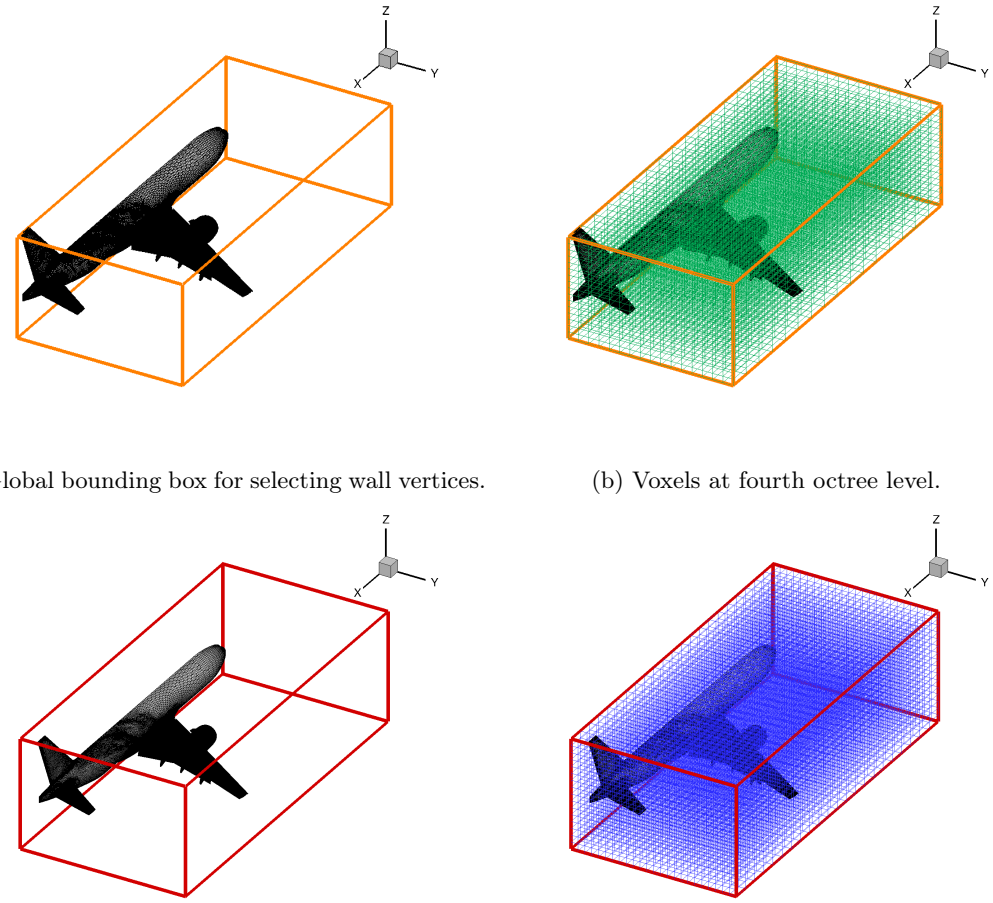
For the computations reported in this paper, each rank holds the entire wall data.

After collecting the wall data, each rank computes a bounding box for each wall face and determines the global largest face lengths in the three coordinate directions. Then, two global bounding boxes are computed. The first box encloses the entire wall surface. The second global box is the first box with the edges extended by *threshold* in the axis directions.

An octree of voxels is constructed for selecting wall vertices that need to be searched. The first global box that tightly bounds wall vertices serves as the octree root node. All branches of the octree have the same depth (the number of levels). Voxels at each octree level represent a partition of the box. In creating the next octree level, voxels of the current level are partitioned into eight voxels. At level m, there are 8^m node voxels. The number of octree levels can be defined by the user; the default number of levels is $\lceil \frac{1}{2} \log_8(N_v) \rceil$, where N_v is the number of wall vertices and $\lceil \cdot \rceil$ is the rounding up (ceil) operation. There are no restrictions on the size of the octree voxels. Smaller voxels may result in a slightly more efficient selection of wall vertices. However, using too small voxels (i.e., too many levels of octree) increases the memory to store the octree. Note that the global octree is a temporary data structure that is used only for the identification of vertices to be searched, and it is not used in the actual wall-distance computations.

The second global box with bounds extended by *threshold* is partitioned into a 3D matrix of voxels. This matrix of voxels is used to select the wall faces to be searched. The voxel size in each direction should be greater than *threshold* and the largest face length in that direction. The matrix dimensions (the number of voxels in each dimension) are chosen based on the limitations of the voxel size. The global box can be adjusted (increased) to accommodate integer number of voxels in each matrix dimension.

Figure 3 illustrates the global bounding boxes, the global matrix of voxels for selecting wall faces, and voxels at the fourth level of the octree for selecting wall vertices. For illustration purposes, the data are computed for grid M of configuration 2.4. (See Table 3.) The size of the first global bounding box is $2533.9 \times 1159.9 \times 710.1$ inches. The fourth level of the octree contains $8^4 = 4,048$ voxels for selecting wall vertices, and the octree voxel size is $158.4 \times 72.5 \times 44.4$ inches. The size of the second global bounding box that has been expanded to accommodate the matrix of voxels is $2708.3 \times 1276.8 \times 851.2$ inches. The global matrix for selecting wall faces contains $22 \times 24 \times 16$ voxels, and the voxel size is $123.1 \times 53.2 \times 53.2$ inches. Recall that, for this configuration and flow (see Section 2), $threshold = 53.2$ inches. Typically the size of the voxels in the matrix and in the octree would be very different. However, for grid M, voxels in the matrix and at the fourth octree level are close to each other. This similarity occurs because the maximum face length in the x-direction is relatively large. On grid G of configuration 2.4, the number of octree levels is still 4. The voxel sizes within the matrix remain limited by threshold, $53.2 \times 53.2 \times 53.2$, and the dimensions of the matrix are $50 \times 24 \times 16$.



(a) Global bounding box for selecting wall vertices.

(b) Voxels at fourth octree level.

(c) Global bounding box for selecting wall faces.

(d) Matrix of voxels.

Figure 3: Global bounding boxes and partitions for Grid M, configuration 2.4.

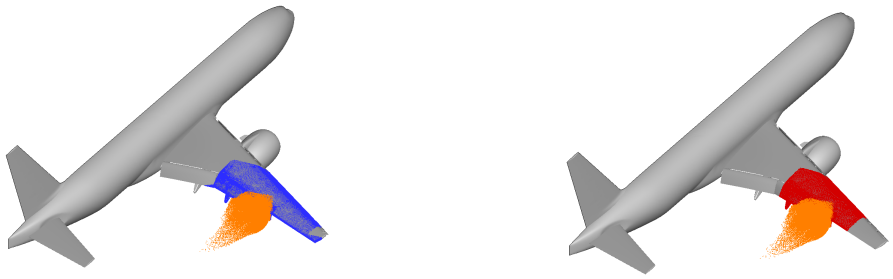
4.2 Selecting Wall Data for Search on Each Rank

Prior to this step, each rank has the selected *threshold* value, the global octree of voxels for selecting wall vertices to be searched, and the global matrix of voxels for selecting wall faces to be searched. At this step, each rank reduces the wall data to be used in wall distance computations. The reduction is accomplished through selection of the wall vertices and wall faces that have a chance to be the closest ones to the local grid points owned by the rank. In the final implementation, only selected wall data will be gathered on each rank. The reduction of the wall data is expected to improve load balance between grid points that are close to the wall and grid points that are far from the wall. Grid points that are beyond *threshold* do not require access to wall faces but may need access to many wall vertices. Grid points that are within *threshold* require access to some wall faces and to a smaller number of wall vertices.

For grid C of configuration 2.4, Fig. 4 illustrates the data used by a specific rank. In the figure, the orange dots are the local grid points owned by the rank, blue dots are the wall vertices that are closest to the local grid points, and the surface is red where the faces are closest to the grid points located within *threshold* from the wall. This rank owns 50,365 grid points. From this total, 43,273 grid points are within *threshold* from the wall. The midspan section of the wing is the closest wall surface to these grid points. For grid points located beyond *threshold*, wall faces are not required. The wall vertices that are closest to these points appear on the midspan and outboard sections the wing.

The following two steps are conducted on each rank to select wall vertices to be searched.

V1 The wall vertices are registered with voxels at the finest octree level. Vertices registered with each voxel are counted and stored. Thus, nonempty voxels are easily identifiable. The information about nonempty boxes propagates to the coarser levels of the octree. Each parent node that has a



(a) Wall vertices (blue dots). (b) Wall faces (red faces).

Figure 4: Wall data on a rank for Grid C, configuration 2.4; orange dots are grid points on the rank.

nonempty child is marked as nonempty.

V2 Using the global octree, the rank identifies the voxels at the finest octree level that host wall vertices selected for the search. The details of this step are given in Section 4.2.1.

Similar steps are conducted to select wall faces.

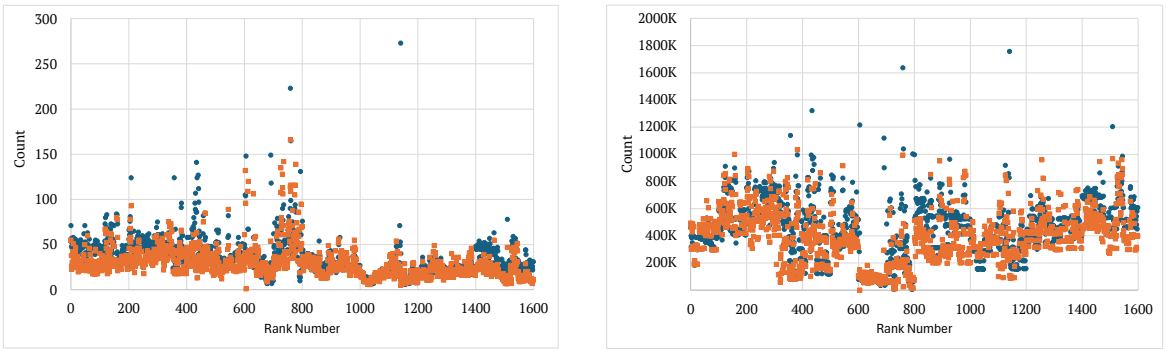
F1 Each rank registers the wall faces with voxels of the global matrix. A face belongs to a voxel if the center of its bounding box is located within the voxel. For each voxel, registered faces are counted and stored. Thus, nonempty voxels are easily identifiable.

F2 Each rank identifies voxels in the global matrix that host faces selected for the search. The details of this step are given in Section 4.2.2.

After these steps, each rank has selected wall vertices and wall faces to compute the wall distance for the grid points owned by the rank. Figure 5 illustrates the distribution of the non-empty voxels that are used for selecting wall vertices and wall faces (Fig. 5(a)) and the actual vertices and faces selected on each rank (Fig. 5(b)). These data are collected for grid F of configuration 2.4 (see Section 2). The computation uses 1600 Central Processing Units (CPU), each with its own rank; the partition size is about 50k grid points per rank. As expected, identified voxels and selected wall data differ significantly between ranks. One interesting observation is that ranks that identify more voxels do not necessarily select more wall data. For example, there are several ranks between 600 and 800 that identify many (over 100) voxels, but they select relatively few wall faces. Overall, the distribution of wall data is relatively balanced; the maximum-to-average ratios are about two and three for wall faces per rank and wall vertices per rank, respectively.

4.2.1 Selecting Wall Vertices

This section describes the steps for selecting wall vertices that can be the closest ones to the grid points on the rank. One approach to achieve the optimal selection requires two passes through the octree of voxels for each local grid point. In the first pass, the shortest maximum distance, $shortest\_max\_dist$, from the point to a nonempty voxel is computed. In the second pass, the wall vertices registered with the finest-level voxels that are closer to the grid point than $shortest\_max\_dist$ are selected. The implemented algorithm saves time by allowing each local grid point to traverse the octree only once. During this traverse, the current, not final, $shortest\_max\_dist$ is used to select wall vertices registered with the voxels at the finest octree level. Some voxels and corresponding vertices may be unnecessarily selected before the true $shortest\_max\_dist$ has been computed. The number of such voxels is assumed small. An additional pass through the identified voxels can be conducted to remove voxels and vertices that have been selected spuriously.



(a) Voxels for wall vertices (dark green dots) and voxels for wall faces (orange dots)

(b) Selected wall vertices (dark green dots) and wall faces (orange dots)

Figure 5: Distribution of wall data for Grid F, configuration 2.4.

Algorithm for selecting wall vertices
Initialize $shortest\_max\_dist = huge$
Set node nd to be the root node of the octree
Procedure VisitNode(pt, nd) /* point pt visits node nd.
  If nd has nonempty children
    Compute the minimum distance, $ch.min\_dist$, from pt to voxels of nonempty children
      /* ch is a child node
    Sort nonempty children in ascending order of $ch.min\_dist$
    Loop over nonempty children in the order
      If $ch.min\_dist \geq shortest\_max\_dist$, Exit Loop **End If**
      VisitNode (pt, ch)
    End Loop
  Else /* nd is a leaf
    Compute the maximum distance, $nd.max\_dist$, from pt to voxel of nd
    $shortest\_max\_dist = min(nd.max\_dist, shortest\_max\_dist)$
    Wall vertices registered with the voxel of nd are selected
  End If
End Procedure VisitNode

4.2.2 Selecting Wall Faces

This section describes steps for selecting wall faces. The selected faces are expected to be closer than *threshold* to at least one local grid point. The algorithm uses the global matrix of voxels and relies on the property that the voxel size in each of the three coordinate directions (x, y, and z) exceeds *threshold* and the largest face length in the corresponding direction. First, an auxiliary box is created for each voxel. This auxiliary box extends the sides of the voxel in each coordinate direction by half of the corresponding largest face length. The algorithm loops over local grid points. It first tries to place the current grid point within a voxel. If the grid point is outside of the voxel matrix, then the point is farther than *threshold* from the wall. For a grid point that falls inside of a voxel, the algorithm checks the emptiness of the host voxel and the surrounding voxels whose auxiliary box is closer to the point than *threshold*. The limitations on the voxel size from below imply that each voxel that satisfies the proximity condition must be either a neighbor or a neighbor of neighbor (in the i, j, k metric) of the host voxel. Wall faces registered with the nonempty voxels that satisfy the proximity condition are selected. If all voxels satisfying the proximity condition are empty, then the grid point is farther from the wall than *threshold*.

4.3 Point-to-Vertex Computations

Prior to this step, each rank has selected the wall vertices that have a chance to be the closest ones for grid points owned by the rank. This section describes the steps for computing the shortest distance from a local grid point to the selected wall vertices.

The number of selected wall vertices on a given rank is denoted as N_v. A bounding box that encloses these vertices is computed. This box serves as the root node of the tree of boxes. If $N_v > 100$, then the root node is not a leaf, and finer levels of the tree are built. Boxes of non-leaf nodes at the current tree level are divided into eight boxes. Wall vertices located within the smaller boxes are counted. Empty boxes are removed. For each nonempty box, a next-level tree node is created, and the parent-child relations are established. The size of the box is adjusted to provide tight bounds for the wall vertices associated with this node. By default, if the number of wall vertices associated with the child node is less than $\sqrt{N_v}$, then the node becomes a leaf of the tree. If the tree level has non-leaf nodes, then the next tree level is generated recursively. At the end of this process, the tree has branches of various lengths. Each tree node has the parent-child relations with preceding and succeeding generations of nodes. Each node knows the number of wall vertices associated with the node (and all descending nodes) and has a tight axis-aligned bounding box that encloses all these wall vertices. Collectively, leaves of the tree contain all wall vertices on the rank, and each leaf has fewer than $\sqrt{N_v}$ wall vertices.

The distance to wall vertices, *wall_dist*, is computed for one grid point at a time. A relative-error tolerance, ϵ, can be introduced to reduce point-to-vertex computations. In particular, if the distance from a point to a node bounding box multiplied by $1 + \epsilon$ is greater than the current wall-distance, then the node and all vertices associated with it are skipped. A recursive procedure is described as follows.

Minimum distance from a point to wall vertices
Initialize *wall_dist* = *huge*
Set *nd* as the root node of the tree
Procedure VisitP2V(*pt*, *nd*) /* point *pt* visits node *nd*.
  If *nd* has children
    Compute the minimum distance, *ch.min_dist*, from *pt* to boxes of children
      /* *ch* is a child node
    Sort child nodes in the ascending order of *ch.min_dist*
    Loop over child nodes in the order
      If $(1+\epsilon)$*ch.min_dist* > *wall_dist*, Exit Loop **End If**
      VisitP2V (*pt*, *ch*)
    End Loop
  Else /* *nd* is a leaf
    Loop over wall vertices associated with *nd*
      Compute distance, *dist*, from *pt* to the wall vertex
      wall_dist = *min*(*wall_dist*, *dist*)
    End Loop
  End If
End Procedure VisitP2V

4.4 Point-to-Face Computations

Computing the distance from a point to a triangular face is significantly more expensive than computing the distance between two points. The following algorithm is used to compute the distance, *dist*, from point $\mathbf{p}$ to triangle $[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$ defined by points $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$. Point $\mathbf{p}_*$ is the point within $[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$ which is the closest one to $\mathbf{p}$; *edist* is an auxiliary value representing the distance from $\mathbf{p}$ to an edge of the triangle.

Minimum distance from a point to a triangular face
$\mathbf{u} = \mathbf{p}_2 - \mathbf{p}_1$, $\mathbf{v} = \mathbf{p}_3 - \mathbf{p}_1$, and $\mathbf{d} = \mathbf{p} - \mathbf{p}_1$
$denom = (\mathbf{u}\cdot\mathbf{u})(\mathbf{v}\cdot\mathbf{v}) - (\mathbf{u}\cdot\mathbf{v})(\mathbf{v}\cdot\mathbf{u})$
$\alpha_1 = \frac{(\mathbf{d}\cdot\mathbf{u})(\mathbf{v}\cdot\mathbf{v}) - (\mathbf{d}\cdot\mathbf{v})(\mathbf{v}\cdot\mathbf{u})}{denom}$ /* normalized projection on $[\mathbf{p}_1, \mathbf{p}_2]$
$\alpha_2 = \frac{(\mathbf{u}\cdot\mathbf{u})(\mathbf{d}\cdot\mathbf{v}) - (\mathbf{u}\cdot\mathbf{v})(\mathbf{d}\cdot\mathbf{u})}{denom}$ /* normalized projection on $[\mathbf{p}_1, \mathbf{p}_3]$
dist = *huge*, *edist* = *huge*
If $(\alpha_1 \geq 0)$ and $(\alpha_2 \geq 0)$ and $((\alpha_1 + \alpha_2) \leq 1)$ /* compute *dist* to the interior of $[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$
  $\mathbf{p}_* = \mathbf{p}_1 + \alpha_1\mathbf{u} + \alpha_2\mathbf{v}$
  $dist = |\mathbf{p} - \mathbf{p}_*|$
Else
  If $(\alpha_1 < 0)$ /* compute *edist* to $[\mathbf{p}_1, \mathbf{p}_3]$
    If $(0 < \alpha_2 < 1)$

```
        p* = p1 + α2 v
      Else If (α2 ≤ 0)
        p* = p1
      Else
        p* = p3
      End If
      edist = |p − p*| , dist = min(edist, dist)
    End If
    If (α2 < 0)                                   /* compute edist to [p1, p2]
      If (0 < α1 < 1)
        p* = p1 + α1 u
      Else If (α1 ≤ 0)
        p* = p1
      Else
        p* = p2
      End If
      edist = |p − p*| , dist = min(edist, dist)
    End If
    If ((α1 + α2) > 1)                            /* compute edist to [p2, p3]
      w = p3 − p2, g = p − p2, α3 = (g·w)/(w·w)   /* normalized projection on [p2, p3]
      If (0 < α3 < 1)
        p* = p2 + α3 w
      Else If (α3 ≤ 0)
        p* = p2
      Else
        p* = p3
      End If
      edist = |p − p*| , dist = min(edist, dist)
    End If
  End If
```

The wall-distance method puts an emphasis on minimizing instances of point-to-face computations. The local matrix of voxels allows quick elimination of grid points that are located farther from the wall than *threshold*. For the grid points that are close to the wall, there are other algorithmic features to reduce point-to-face computations. In particular, each face is wrapped in an axis-aligned bounding box. Even if the algorithm logic indicates the need to compute the distance from a point to a face, the distance is first computed to the face bounding box. This preliminary computation may reveal that the distance from the point to the face bounding box is greater than the current wall-distance estimate. It means that the face is too far from the point, and the point-to-face computation is not needed. Also, a relative-error tolerance, ϵ, can be been introduced to further reduce point-to-face computations. In particular, if the distance from a point to a face bounding box multiplied by $(1 + \epsilon)$ is greater than the current wall-distance estimate for the point, then the distance from the point to the face is not computed.

The following steps construct data structures for computing the point-to-face wall distance.

- The largest x-, y-, and z-directional lengths among the selected faces are computed.
- The minimum local voxel size in each direction is determined to be the minimum of *threshold* and the largest selected-face length in that direction.
- A bounding box enclosing all selected wall faces on the rank is computed and extended by *threshold* in each coordinate direction.
- The extended bounding box is divided into the matrix of voxels. The voxel directional size is the minimum local voxel size in the corresponding direction. The extended bounding box can be adjusted to accommodate reasonable dimensions for the matrix of voxels.
- Each selected face on the rank is registered with a voxel.
- The bounding box of each voxel is adjusted to provide tight bounds for the faces registered with this voxel. This adjustment may result in shrinking or expanding the original voxel bounds. The

bounding boxes of neighboring voxels can now overlap. Because the voxel size exceeds the largest face length, the adjusted bounding box can reach only to neighbors of the current voxel.

- If there are more than 50 faces registered with a voxel, then the voxel is considered as overpopulated. A tree data structure is created to search among faces of overpopulated voxels.
- The tree data structure for overpopulated voxels is similar to the one described in Section 4.3. The main differences are the following. The tree is built for faces that are associated with a specific voxel. The maximum number of the tree levels is specified as 10. A tree node is declared as a leaf if it has fewer than 50 faces. The node's bounding boxes are adjusted to accommodate faces associated with the tree node and can overlap.

The algorithm to improve the wall-distance estimate for the grid points located near the wall is as follows. For each local grid point, the wall distance, $wall\_dist$, is initialized by the distance to the wall vertices (see Section 4.3). For local points that are closer than $threshold$ to the wall, the algorithm improves this estimate by computing the distance to wall faces. The following steps are conducted for one grid point, pt, at a time, where av is an anisotropic voxel within the local voxel matrix, fc is a face, and fcb is the bounding box of fc.

Minimum distance to wall face

If pt is inside voxel matrix /* otherwise, pt is beyond $threshold$
  Find av that contains pt
  Loop over nonempty neighbor and neighbor-of-neighbor voxels
    Compute distance, $av.dist$, from pt to bounding box of av
    If $av.dist > threshold$ **or** $(1+\epsilon)av.dist > wall\_dist$, Exit Loop **End If**
    If av has more than 50 faces /* av overpopulated; recursive tree call
      VisitP2F (pt, av)
    Else /* check all faces of av
      Loop over faces fc associated with av
        Compute distance, $fcb.dist$, from pt to bounding box of fc
        If $fcb.dist > threshold$ **or** $(1+\epsilon)fcb.dist > wall\_dist$, Cycle **End If**
        Compute distance, $fc.dist$, from pt to fc
        $wall\_dist = min(wall\_dist, fc.dist)$
      End Loop
    End If
  End Loop
End If

Recursive search within overpopulated voxels

Procedure VisitP2F(pt, nd) /* point pt visits node nd.
  If nd has children
    Compute distance, $ch.dist$, from pt to the bounding boxes of children /* ch is a child node
    Sort child nodes in the ascending order of $ch.dist$
    Loop over child nodes in the order
      If $ch.dist > threshold$ **or** $(1+\epsilon)ch.dist > wall\_dist$, Exit Loop **End If**
      VisitP2F (pt, ch)
    End Loop
  Else /* nd is a leaf
    Loop over faces fc associated with nd
      Compute distance, $fcb.dist$, from pt to the bounding boxes of fc
      If $fc.dist > threshold$ **or** $(1+\epsilon)fcb.dist > wall\_dist$, Cycle **End If**
      Compute distance, $fc.dist$, from pt to fc
      $wall\_dist = min(wall\_dist, fc.dist)$
    End Loop
  End If
End Procedure VisitP2F

5 Numerical Results

5.1 Comparison with Reference Methods

The wall distance has been computed for four grids generated for HLPW5 configuration 2.4 (Table 3) by the three methods: the new method, the FUN3D legacy method, and the *refine* method. The new method uses $threshold = 53.2$ inches and zero relative-error tolerance $\epsilon = 0$. The reference methods are applied without any additional optimization. All methods use the same dual-socket Intel Xeon Gold 6148 Skylake compute nodes consisting of 40 2.40 GHz CPU cores. One MPI rank is used per CPU core, and nodes are connected with Infiniband. The grid partitions target about 50 thousand grid points per CPU core on each grid. The new method and the legacy FUN3D method use the same grid partitions. The *refine* method uses its own partitioner. The wall-distance performance on different grid partitions can vary significantly. A partition tuned for the wall-distance calculations would evenly distribute grid points that are close to the wall. However, since grids are partitioned to equidistribute grid points while minimizing edge cuts, redistribution of grid points is considered impractical. Table 4 shows the total time taken by each wall-distance method. The new method is significantly faster than the reference methods, especially on fine grids. On grid C with six million grid points, the new method and the legacy method show comparable performance; the *refine* method is almost four times slower. The speedup provided by the new method is further increased on finer grids. On grid G with 178 million grid points, the new method provides a 3X speedup over the legacy method and a 5X speedup over the *refine* method.

Table 4: Performance of methods for wall-distance computation

Grid	Grid points (millions)	CPU cores	New method time (s)	Legacy method time (s)	*refine* method time (s)
C	6	120	1.36	1.96	5.80
M	15	320	1.91	4.05	8.38
F	81	1600	5.59	15.46	26.78
G	178	3560	9.45	30.12	51.77

Although scalability of the new method is better than that of the reference methods, none of the considered methods is scalable. The time for wall-distance computations increases multifold on fine grids. A scalable method would show a constant time, given that the size of the partitions remains approximately the same for all grids. One reason for the poor scalability is the fact that each rank gathers the entire wall data.

The effect of the new wall-distance method on solution accuracy is assessed for configuration 2.1 and the following freestream flow conditions: angle of attack is $\alpha = 6^\circ$, Mach number is $M = 0.2$, Reynolds number is $Re_{MAC} = 5.4 \times 10^6$, the reference static temperature $T_{ref} = 518.67$ °R, and the reference static pressure is $P_{ref} = 14.696$ psi. The RANS equations use the negative version of the Spalart-Allmaras turbulence model [34]. FUN3D converged residuals to machine-zero levels on grid M (see Table 2) for the solutions corresponding to the three different wall-distance methods. The errors in the solutions are assessed by subtracting the non-dimensional solution quantities computed with the exact wall distance provided by *refine* from the corresponding solution quantities computed with the wall distance provided by either the new method or the legacy method. Table 5 compares the maximum and root-mean-square (RMS) norms of the wall-distance errors for all grid points. In parentheses, the values of the exact and calculated wall distance are shown for the grid point where the maximum error is observed. The maximum and RMS norms of the errors are similar for the new method and the legacy method. However, the maximum error in the legacy method occurs for a grid point close to the wall, whereas the maximum error in the new method occurs at a grid point far from the wall. Table 6 shows a comparison of errors in the eddy viscosity, which are six orders of magnitude smaller with the new method compared with the legacy method. This difference is also present in the error assessments for the pressure and skin-friction coefficients shown in Table 7. However, the absolute errors are small in the solutions computed with either method. Finally, lift, drag, and moment coefficients are compared in Table 8. The coefficients computed with the new method and with the exact wall distance are identical. The coefficients computed with the legacy method match to 5-6 significant digits. Differences are highlighted in bold font in Table 8.

Table 5: Wall-distance error

Norm	New method	Legacy method
Max	3.08	2.99
	(exact: 158.50, calculated: 161.58)	(exact: 0.06, calculated: 3.05)
RMS	0.02	0.01

Table 6: Eddy viscosity error

Norm	New method	Legacy method
Max	6.88E-04	148.33
RMS	1.83E-06	0.59

Table 7: Error in pressure and skin friction coefficients

Norm	New method	Legacy method
pressure: Max	0	7.91E-05
pressure: RMS	0	8.58E-07
skin friction: Max	1.86E-09	3.28E-03
skin friction: RMS	1.11E-11	1.92E-05

Table 8: Aerodynamic coefficients

Coefficient	New method	Legacy method	*refine* method
C_L	6.934988E-01	6.93**5006**E-01	6.934988E-01
C_D	3.932581E-02	3.932**855**E-02	3.932581E-02
CM_x	3.027655E+02	3.027**61**E+02	3.027655E+02
CM_y	-1.378081E-01	-1.378**58**E-01	-1.378081E-01
CM_z	1.823076E+01	1.823**041**E+01	1.823076E+01

5.2 Profiling Steps of New method

There are four major steps comprising the new wall-distance method listed in Section 4. Table 9 provides a breakdown of the time that the new method spends on these four steps on grids M and F of configuration 2.4. Grid M has 15 million grid points, 540 thousand wall vertices, and 1.1 million wall faces and uses 320 CPU cores. Grid F has 82 million grid points, 1.2 million wall vertices, and 3.7 million wall faces and uses 1600 CPU cores. (See Tables 3 and 4.) For each step of the method, the minimum, maximum, and average time are shown, as well as the specific times for the fastest and slowest ranks. The total time used by different ranks ranges from 1 to 1.9 seconds on grid M and from 3 to 5.6 seconds on grid F. For reference, the time spent by the solver on a simple nonlinear iteration is approximately 2 seconds on grid M and 2.17 seconds on grid F. The time spent by the new wall-distance method on Step 1 (gathering the global wall data on a rank) is practically the same for all ranks of a given grid. It is currently approximately 34% and 42% of the maximum total time on grids M and F, respectively. This step exhibits the worst scaling. The time spent at this step is expected to be reduced significantly when only the necessary wall data are collected on each rank. At Step 2 (selecting the local wall data), BVH data structures are used to select the wall vertices and faces that can be the closest ones for the local grid points. This step is currently not needed for gathering data, but it significantly improves performance of the actual wall-distance computations at Step 3 and Step 4. Step 2 currently scales with the number of wall faces. Steps 3 and 4 perform the point-to-vertex and point-to-face computations, respectively. Collectively, they take approximately 40-50% of the time on the slowest ranks and 15-30% of the time on average ranks. The point-to-vertex computations scale well relative to other steps; the time of Step 3 increased by 47% on grid F compared to the time on grid M. The point-to-face computations take significantly more time than the point-to-vertex computations and currently scale with the number of wall faces. The slowest rank on grid F searches 757 thousand wall vertices and 684 thousand wall faces identified as potentially the closest ones for the grid points on the rank. An average rank searches 460 thousand vertices and 380 thousand faces. The number of wall faces searched by the slowest rank is among the higher ones, and the number of searched wall vertices is also high. The fastest ranks spend little time on point-to-vertex and point-to-face computations; most of their time is spent on gathering the wall data.

Table 9: Breakdown of wall-distance computation time

	Gathering global wall data (s)	Selecting local wall data (s)	Point-to-vertex computations (s)	Point-to-face computations (s)	Total wall distance (s)
			Grid M		
Min	0.62	0.17	0.035	0.013	0.99
Max	0.66	0.44	0.46	0.73	1.91
Average	0.63	0.31	0.094	0.26	1.30
Fastest rank	0.66	0.21	0.11	0.013	0.99
Slowest rank	0.65	0.33	0.36	0.57	1.91
			Grid F		
Min	2.32	0.47	0.04	0.0	3.03
Max	2.41	0.97	1.38	1.87	5.59
Average	2.36	0.73	0.16	0.38	3.64
Fastest rank	2.36	0.48	0.039	0.15	3.03
Slowest rank	2.36	0.90	0.53	1.80	5.59

The fraction of time spent on MPI communication is relatively low. Table 10 lists the MPI time on grids M and F for collecting the surface, vertex, and face data of the global wall on each rank. During the wall-surface MPI communication, each rank collects indices of wall vertices. During the MPI communication for vertices, each rank collects coordinates of wall vertices. During the MPI communication for faces, each rank collects indices of wall faces and links faces to vertices. The entire MPI time is 9-12% of the total time on the slowest rank; MPI communication accounts for 26-30% of the time to gather the global wall data.

Table 10: MPI time for collecting global wall data

	Wall surface (s)	Wall vertices (s)	Wall faces (s)	New method total time (s)
		Grid M		
Min	0.05	0.040	0.095	0.99
Max	0.07	0.042	0.12	1.91
Average	0.06	0.041	0.11	1.30
		Grid F		
Min	0.19	0.15	0.29	3.03
Max	0.22	0.16	0.31	5.59
Average	0.21	0.15	0.30	3.64

5.3 Effect of *threshold* and Relative-Error Tolerance

Calculation of the distance from a point to a triangular face (see Algorithm in Section 4.4) is the most expensive calculation in evaluating the wall distance for a grid point. BVH data structures eliminate large portions of the domain from consideration for the point-to-face calculations. The new wall-distance method provides two additional parameters to further minimize the instances of direct point-to-face calculations: a grid-point separator, *threshold*, and a relative-error tolerance, ϵ. For the grid points that are closer to the wall than *threshold*, the wall distance is computed to wall faces; for the points that are farther from the wall than *threshold*, the wall distance is computed to wall vertices. With $\epsilon = 0$, the point-to-vertex and point-to-face calculations are performed for any vertex and face whose bounding box indicates that the current wall-distance estimate can be potentially reduced. With $\epsilon > 0$, if the distance to a bounding box indicates that the vertices or faces associated with the box cannot result in a wall-distance estimate that is smaller than the most recent estimate divided by $(1+\epsilon)$, then distances to vertices and faces of this box are not computed. Decreasing *threshold* or increasing ϵ reduces the direct point-to-face calculations in wall-distance computations. In the performance assessments reported in the previous sections, *threshold* = 53.2 inches and $\epsilon = 0$ have been used for configuration 2.4. This *threshold* value is based on doubling the boundary-layer thickness estimate, Eq. 1. In this section, the performance of wall-distance computations are assessed for solutions computed with baseline *threshold* = 53.2 and

$\epsilon = 0.02$ and $\epsilon = 0.05$ along with $threshold = 26.6$.

Table 11 shows the total time for computing the wall distance with four $threshold/\epsilon$ combinations. The combination 53.2/0 represents the baseline wall-distance computations. Increasing the relative-error tolerance results in a noticeable speedup on all grids: $\epsilon = 0.02$ results in 1.10X speedup on grid C and 1.21X speedup on grid G; $\epsilon = 0.05$ results in 1.18X speedup on grid C and 1.30X speedup on grid G. The speedup increases on finer grids. Reduction of *threshold* results in 1.07X speedup on grid C and 1.16X speedup on grid G. Different combinations of ϵ and *threshold* will be tested in the future.

Table 11: Parametric study of performance

Grid	Grid points (millions)	CPU cores	*threshold* (inches)/ϵ			
			53.2/0	53.2/0.02	53.2/0.05	26.6/0
C	6	120	1.46	1.33	1.24	1.36
M	15	320	1.91	1.75	1.61	1.81
F	81	1,600	5.59	4.64	4.21	4.83
G	178	3,560	9.45	7.84	7.27	8.17

The effects of *threshold* and ϵ on the accuracy of CFD solutions are studied for configuration 2.1 and flow conditions described in Section 5.1. The new method uses $threshold = 54.2$ inches with $\epsilon = 0, \epsilon = 0.02$, and $\epsilon = 0.05$ as well as $threshold = 27.1$ inches with $\epsilon = 0$. Solutions with machine-zero levels of residuals have been computed on grid M. The wall-distance errors are evaluated in Table 12 for $\epsilon = 0$. With $threshold = 27.1$, the maximum and RMS norms of the wall-distance error increase by 32% and 40% respectively. The maximum error occurs closer to the wall than the maximum error in computations with $threshold = 54.2$. The following detailed analysis of the solution accuracy shows that the reduction of *threshold* has a negligible effect on solution accuracy.

Table 12: Parametric study of wall-distance error

Norm	*threshold* (inches)/ϵ	
	54.2/0	27.1/0
Max	3.1	4.1
	(exact: 158.5; calculated: 161.6)	(exact: 34.1; calculated: 38.2)
RMS	0.015	0.021

Table 13 compares eddy-viscosity errors in solutions computed with the four $threshold/\epsilon$ combinations. The effect of *threshold* reduction on eddy-viscosity errors is negligible, with the maximum increase from 0.001% to 0.005%. Increasing the relative-error tolerance has a more significant effect: the absolute error increases by several orders of magnitude, and the maximum relative-error increases to 16% and 47% for $\epsilon = 0.02$ and $\epsilon = 0.05$, respectively. However, the overall error in the eddy viscosity remains low. For reference, the maximum eddy viscosity observed in the solution with $threshold = 54.2$ inches and $\epsilon = 0$ is 1235.14.

Table 13: Parametric study of eddy-viscosity error

Norm	*threshold* (inches)/ϵ			
	54.2/0	54.2/0.02	54.2/0.05	27.1/0
Max	6.9E-04	7.6E-01	1.9	1.3E-03
Max(%)	0.001%	15.5%	47.1%	0.005%
RMS	1.8E-06	1.8E-02	3.4E-02	8.8E-05

Table 14 compares errors in the surface pressure and skin friction. Again, the effect of *threshold* reduction is negligible. Although increasing the value of ϵ increased the error norms by 2-3 orders of magnitude for the pressure and by 5-6 orders magnitude for the skin friction, the overall errors remain small, a fraction of a percent.

Finally, Table 15 shows the effect of the method tuning parameters on the lift, drag, and moment coefficients. Bold font highlights the digits that do not match the corresponding digits in the coefficients computed from the baseline solution. The coefficients computed with the reduced *threshold* are identical to the coefficients computed from the baseline solution. Increasing the relative-error tolerance results in a mismatch in the sixth significant digit for the drag coefficient and in the seventh significant digit for the

Table 14: Parametric study of errors in pressure and skin friction coefficients

Norm	*threshold* (inches)/ϵ			
	54.2/0	54.2/0.02	54.2/0.05	27.1/0
pressure: Max	0	3.99E-06	2.26E-05	5.96E-08
pressure: RMS	0	5.91E-08	2.37E-07	3.05E-10
skin friction: Max	1.86E-09	1.19E-03	1.19E-03	1.86E-09
skin friction: RMS	1.11E-11	1.96E-06	2.08E-06	5.73E-11

lift and moment coefficients. More studies are needed to establish optimal values for the relative-error tolerance and *threshold*.

Table 15: Parametric study of aerodynamic coefficients

	threshold (inches)/ϵ			
	54.2/0	54.2/0.02	54.2/0.05	27.1/0
C_L	6.934988E-01	6.934987E-01	6.934990E-01	6.934988E-01
C_D	3.932581E-02	3.932613E-02	3.932630E-02	3.932581E-02
CM_x	3.027655E+02	3.027654E+02	3.027655E+02	3.027655E+02
CM_y	-1.378081E-01	-1.378078E-01	-1.378073E-01	-1.378081E-01
CM_z	1.823076E+01	1.823074E+01	1.823073E+01	1.823076E+01

6 Summary and Future Work

Wall distance is a fundamental quantity for many Reynolds-averaged Navier-Stokes (RANS) turbulence models. The efficiency of wall-distance calculations has become more critical for computational fluid dynamics (CFD) applications as the size of computational grids has significantly increased in recent years. Wall-distance methods implemented in modern high-performance CFD solvers have often been developed for static grids, follow nearest-neighbor search algorithms developed for applications in other areas of science and engineering, and may not be well-suited for high-performance CFD computing. This paper has presented a new search-based wall-distance method that is tailored for the specific requirements of turbulent-flow simulations on general unstructured grids. The method uses native partitions of the flow solver, limits searches on each rank to the wall data (vertices and faces) that could possibly be the closest ones to the grid points on the rank, and applies different accuracy criteria for points that are near and far from the wall. The new method has been implemented in a NASA unstructured-grid, high-performance CFD solver, FUN3D. Tests have been completed with an initial implementation, but further enhancements and validation are in progress.

The new method strives to minimize calculating the distance between a grid point and a wall face, which is the most expensive operation. One of the main features of the new method is a user-defined parameter *threshold* that separates grid points into two groups. For grid points that are closer to the wall than *threshold*, the minimum distance is computed to wall faces; for grid points that are beyond *threshold*, the wall distance is computed to wall vertices. The point-to-vertex distance calculation is significantly less expensive than the point-to-face distance calculation. Bounding-volume hierarchy (BVH) approaches have been successfully used in many nearest-neighbor algorithms to exclude large portions of computational domain from the search. BVH data structures based on a tree of bounding boxes and a matrix of voxels have been employed to select the minimum required wall data for each rank and to accelerate the search for the closest wall vertices and faces. The new wall-distance method also provides another parameter, the relative-error tolerance, which, when greater than zero, can further reduce the number of point-to-vertex and point-to-face computations.

The performance and accuracy of the method have been favorably compared with that of two established reference wall-distance methods. The reference methods are the legacy method in FUN3D and a method implemented in the NASA mesh-adaptation tool, *refine*. The assessments have been performed for unstructured, mixed-element grids generated for two configurations used in the Fifth AIAA CFD High-Lift Prediction Workshop (HLPW5). The new method outperforms the reference methods by factors of 3 to 5; the speedup increases on finer grids. Accuracy assessments have been carried out for solutions computed with the negative variant of the Spalart-Allmaras model for a set of HLPW5 flow

conditions on a representative grid. Wall distance, eddy viscosity, surface pressure, skin friction, and aerodynamic coefficients computed with the wall distance provided by the current method have been compared with those computed with the exact wall distance. The differences have been found negligible; specifically, the lift, drag, and three moment coefficients match to seven significant digits.

The four major steps of the new method, namely, (1) gathering the global wall data, (2) selecting the local wall data, (3) point-to-vertex computations, and (4) point-to-face computations, have been profiled on two grids. The first step still uses a legacy approach in which each rank gathers the entire wall surface. This step takes the largest portion of the total wall-distance time and scales the worst with problem size among the four steps. Replacing this step with an approach in which each rank gathers only the necessary wall data is the immediate task for future work. Other steps take less time and scale somewhat better, although none of the steps scales perfectly, which is expected for partitions that are not load-balanced for the wall-distance computations.

Parametric studies have been conducted to assess the impact of varying *threshold* and the relative-error tolerance on performance and accuracy. Increasing the relative-error tolerance to 2% or 5% resulted in an additional noticeable speedup on all grids ranging from 1.1X to 1.3X. The speedup increases on fine grids. Reducing *threshold* by a factor of two resulted in a speedup ranging from 1.07X on a coarse grid to 1.16X on a finer grid. The impact of these variations on solution accuracy is minimal. The largest error observed for the 5% tolerance occurred in the sixth significant digit of the drag coefficient.

Future work to further improve wall-distance computations for turbulent-flow simulations will focus on the following steps:

1. Replace gathering the entire wall surface on each rank with collecting only necessary wall vertices and faces
2. Study ways to further reduce instances of point-to-face computations, e.g., by using oriented bounding boxes.
3. Assess performance of the wall-distance method for unsteady flow simulations on dynamic, deforming, overset, unstructured grids.
4. Study approaches to implement of this method on GPU architectures.

Acknowledgments

The Transformational Tools and Technologies (TTT) project of the NASA Transformative Aeronautics Concepts Program (TACP) partially funded the work reported herein. The first author was supported by the NASA Langley Research, Science, and Engineering Services (RSES) contract with Analytical Mechanics Associates. All computations have been performed on the NASA Langley K cluster. The authors thank Yi Liu (NASA Langley) for providing assistance with the FUN3D solver and HLPW5 solutions. Gratitude is also expressed to William Jones, Gabriel Nastac, Mark Lohry, Kyle Anderson, Emmett Padway, Kevin Jacobson, and Josh Wagner (NASA Langley) for attending bi-weekly brainstorming sessions and providing ideas and encouragement for this work.

References

[1] M. R. Abbasifard, B. Ghahremani, and H. Naderi. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 95(25):39–52, 2014.

[2] A. Guezlec. "Meshsweeper": dynamic point-to-polygonal mesh distance and applications. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):47–61, 2001.

[3] S. Auer and R. Westermann. A semi-Lagrangian closest point method for deforming surfaces. *Computer Graphics Forum*, 32:207–214, 2013.

[4] I. Wald, W. Usher, N. Morrical, L. Lediaev, and V. Pascucci. RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location. In *Conference on High-Performance Graphics*, 7–13, 2019, Strasbourg, France

[5] B. Roget and J. Sitaraman. Wall distance search algorithm using voxelized marching spheres. *Journal of Computational Physics*, 241:76–94, 2013.

[6] E. van der Weide, G. Kalitzin, J. Schluter, and J. J. Alonso. Unsteady Turbomachinery Computations Using Massively Parallel Platforms. AIAA Paper 2006-0421, 2006.

[7] A.D. Boger. Efficient Method for Calculating Wall Proximity. *AIAA Journal*, 9(12):2404–2406, 2001.

[8] J. A. Sethian. Fast Marching Methods. *SIAM review*, 41(2):199–235, 1999.
[9] C. Zong, J. Xu, J. Song, S. Chen, S. Xin, W. Wang, and C. Tu. P2M: A Fast Solver for Querying Distance from Point to Mesh Surface. *ACM Transaction on Graphics*, 42(4), No.147, 2023.
[10] P.G. Tucker. Differential equation-based wall distance computation for DES and RANS. *Journal of Computational Physics*, 190(1):229–248, 2003.
[11] P. Tucker, R. Bartels, C. Rumsey, P. Spalart, and R. Biedron. Computations of Wall Distances Based on Differential Equations. AIAA Paper 2004-2232, 2004.
[12] J. Xu, C. Yan, and J. Fan. Computations of Wall Distances by Solving a Transport Equation. *Applied Mathematics and Mechanics*, 32(2):141–150, 2011.
[13] R. Loehner, D. Sharov, H. Luo, and R. Ramamurti. Overlapping unstructured grids. AIAA Paper 2001-0439, 2001.
[14] L. Devroye, C. Lemaire, and J.-M. Moreau. Expected time analysis for Delaunay point location. In *Computational Geometry*, 29(2):61–89, 2004.
[15] M. Sharifzadeh and C. Shahabi. VoR-Tree: R-Trees with Voronoi Diagrams for Efficient Processing of Spatial Nearest Neighbor Queries. In *Proc. VLDB Endow.*, 3(1-2):1231—1242, 2010.
[16] I. Wald. Fast construction of SAH BVHs on the Intel Many Integrated Core (MIC) architecture. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):47–57, 2012.
[17] D. Meister and J. Bittner. Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1345–1353, 2018.
[18] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, 322–331, 1990.
[19] T. Larsson and T. Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computers & Graphics*, 30(3):450–459, 2006.
[20] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.
[21] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14:105–116, 1995.
[22] L. Kavan and J. Žára. Fast collision detection for skeletally deformable models. *Computer Graphics Forum*, 24:363–372, 2005.
[23] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 171–180, 1996.
[24] J. T. Klosowski, M. Held, J. SB Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.
[25] W. K. Anderson et. al. *FUN3D Manual: 14.0.2.* NASA TM 20230004211, November 2023.
[26] L. Wigton. Research in computational aeroscience applications implemented on advanced parallel computing systems. Technical report, NASA CR-96-206062, 1996.
[27] M. A. Park, N. Barral, D. Ibanez, D. S. Kamenetskiy, J. A. Krakos, T. R. Michal, and A. Loseille. Unstructured Grid Adaptation and Solver Technology for Turbulent Flows. AIAA Paper 2018-1103, 2018.
[28] The 5th AIAA CFD High Lift Prediction Workshop Website. https://hiliftpw.larc.nasa.gov/index.html, 2023.
[29] D. Lacy and A. M. Clarck. Definition of Initial Landing and Takeoff Reference Configurations for the High Lift Common Research Model (CRM-HL). AIAA Paper 2020-2771, 2020.
[30] H. Schlichting. Boundary-Layer Theory, 7th ed. McGraw Hill, New York, USA, 1979.
[31] HeldenMesh. https://heldenaero.com/heldenmesh
[32] L. Andrews. A Template for the Nearest Neighbor Problem. *C/C++ Users Journal*, 19(11):40-49, 2001.
[33] D. Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition. *Addison-Wesley*. Section 6.2.3: Balanced Trees, pp.458–481, 1998.
[34] S. R. Allmaras, F. T. Johnson, and P. R. Spalart. Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model. *Seventh International Conference on Computational Fluid Dynamics*, ICCFD7-1902, Big Island, HI, 2012.