Development of a Cartesian Cut-Cell Solver for Viscous Flows

Alex Kleb* and Krzysztof J. Fidkowski† and Joaquim R. R. A. Martins‡
University of Michigan, Ann Arbor, Michigan 48019

Mesh generation in Computational Fluid Dynamics (CFD) is one of the most critical and time consuming parts of the analysis process. Cartesian cut-cell methods with adjoint based adaptive mesh refinement (AMR) have been used on inviscid problems to automate the meshing procedure for arbitrary geometries. However, these methods struggle with viscous equations because they lack a robust way to account for varying scales in spatial dimensions. A Viscous Aerodynamic Cartesian Cut-cell (VACC) toolbox has been developed to facilitate Cartesian cut-cell research on high Reynolds number viscous flows. This paper outlines the progress made so far on the foundational mesh generator and flow solver.

I. Introduction

A key part of Computational Fluid Dynamics (CFD) is generating the computational mesh where the governing flow equations are solved. Finding the appropriate distribution of mesh points within the domain for balancing cost and accuracy can be extremely challenging, sometimes taking weeks or months. Additionally, when performing numerical aerodynamic shape optimizations (ASO), the optimizer may provide infeasible intermediate design parameters that break a naive mesh generation or warping procedure [1]. In a study done by Seraj and Martins, [2] a supersonic configuration took two months of tweaking parameters to obtain a mesh that converged a flow solution and allowed for ASO. While CFD methods have become more efficient, the mesh generation process for complex geometries remains the most time intensive part of the solution procedure [3].

One promising method for generating meshes automatically is adaptive Cartesian cut-cells [4]. To generate a mesh, an arbitrarily complex geometry is set into and cut out of a background Cartesian mesh [5]. The Cartesian cells can be split based on geometric and adjoint based error metrics to reduce the discretization error. This produces an accurate solution with minimal user input and a priori assumptions about flow structure.

However, automatic methods are currently only viable for inviscid or incompressible flows [6–9]. When performing high fidelity analysis, viscous effects such as skin friction can contribute significantly to drag calculations and aircraft design. The main challenge for Cartesian cut-cell methods is resolving boundary layers [9]. The addition of higher order derivatives in the Navier-Stokes equations results in numerical difficulties near the boundary. With a naive approach, an intractable number of isotropic Cartesian cells would be needed near the body to accurately resolve the boundary layer evolution. Typical boundary conforming anisotropic meshing techniques get around this by stretching cells along the surface to avoid wasting grid points parallel to the body. The robust and automatic nature of the Cartesian cut-cell method would be squandered if cells needed to be aligned with the flow direction, making this approach implausible.

There are a couple proposed solutions to the problem of viscous cut-cells in the literature, but they often give up accuracy or robustness in the process. The first is the strand grid or near-body mesh approach [10]. This method takes an initial geometry and creates a fitted near-body mesh similar to typical unstructured meshing techniques. After the near-body mesh has been generated, it can be cut out of a Cartesian background mesh in the same way as before. In most cases, generating a good near-body mesh for resolving the boundary layer is the most challenging part of the meshing procedure and sacrificing robustness here can be costly. A different approach uses wall functions to approximate the behavior of the boundary layer in cells close to the body [11]. This method retains robustness by using specific rules about determining forcing point locations based on cut-cell centroids and wall normals. However, using an approximation for the boundary layer means that the accuracy can break down when the forcing point is outside certain regions of the boundary layer. When a configuration has a boundary layer with varying thickness, picking the correct location for accurate forcing points can be impossible without knowing the solution a priori.

The most promising solution thus far is an approach proposed by Berger and Aftosmis [12]. The approach is similar to the wall function. Rather than only applying the wall function at a single forcing point, linelets protruding from the body provide a domain to solve a one dimensional ordinary differential equation (ODE) to approximate the boundary

*PhD. Candidate, Department of Aerospace Engineering. Student Member AIAA.
†Professor, Aerospace Engineering Department. Associate Fellow AIAA.
‡Professor, Aerospace Engineering Department. Fellow AIAA.

layer. To provide a base for further research, we have built a new Cartesian cut-cell mesh generator and flow solver: Viscous Aerodynamic Cartesian Cut-cells (VACC).

The goal of this paper is to outline the progress made on VACC and discuss tricks used for speeding up the convergence of the explicit time marching scheme. In Sec. II we briefly outline the mesh generation procedure. Sec. III presents our second-order formulation of inviscid fluxes. These processes are extended in Sec. IV to account for the viscous fluxes. Finally, we discuss convergence acceleration for our explicit time marching scheme in Sec. V.

II. Mesh Generation

VACC creates 2D Cartesian cut-cell meshes for wetted surface geometries within a square domain. Geometries consist of multiple piecewise linear wetted surfaces. The mesh generator can handle degenerate geometries and identify arbitrary numbers of split cells as unique compute volumes. A discussion of degenerate geometries is outside the scope of this paper. This section will assume there are no degeneracies when describing the mesh generation procedure. Figure 1 presents this generic form of the mesh generation procedure. VACC starts with a single rectangular domain

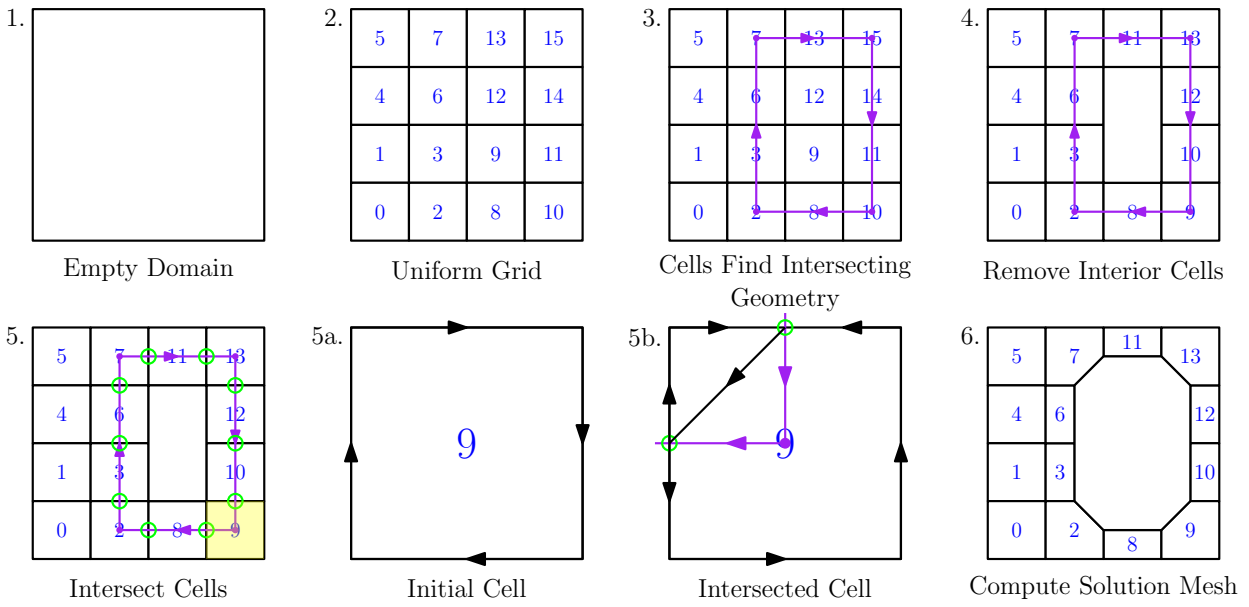


Fig. 1 The mesh generation procedure starts with an empty domain. Cut cells are cut by drawing straight lines between entrance and exit points. After intersecting geometric cells, compute volumes are identified by counter clockwise loops.

that is subdivided into smaller cells as the grid is refined. Before intersecting any geometry, a uniform grid is placed in the domain. Cells use an alternating digital tree (ADT) to identify potential geometry element intersections [13]. This provides a list of candidate elements that have bounding boxes that intersect the bounding box of the cells.

Once a list of candidate elements is found, each element can be checked for intersection by comparing against the cell bounds. Since these cells are located on an exact Cartesian grid, a unique approach to identifying intersections can be taken. Figure 2 presents a few different cases for classifying these elements as intersections or not. The trivial cases are considered by performing binary operations on 4 bit codes assigned to element end points. A point is assigned a code based on its relationship with the edges of Cartesian cells, which are represented exactly in floating point arithmetic. The first bit indicates if the point has a lower x value than the lower cell boundary. The second bit indicates if the point has a higher x value than the higher cell boundary. The final two bits represent the same idea for the y direction.

Element A would never be provided as a candidate intersection because its bounding box does not intersect the cell at all. This corresponds to,

$$P_{A,1} \,\&\, P_{A,2} \neq 0,$$

where & is the bitwise and operator and $P_{A,1}$ and $P_{A,2}$ are out-codes for the two endpoints for element A. Element B is

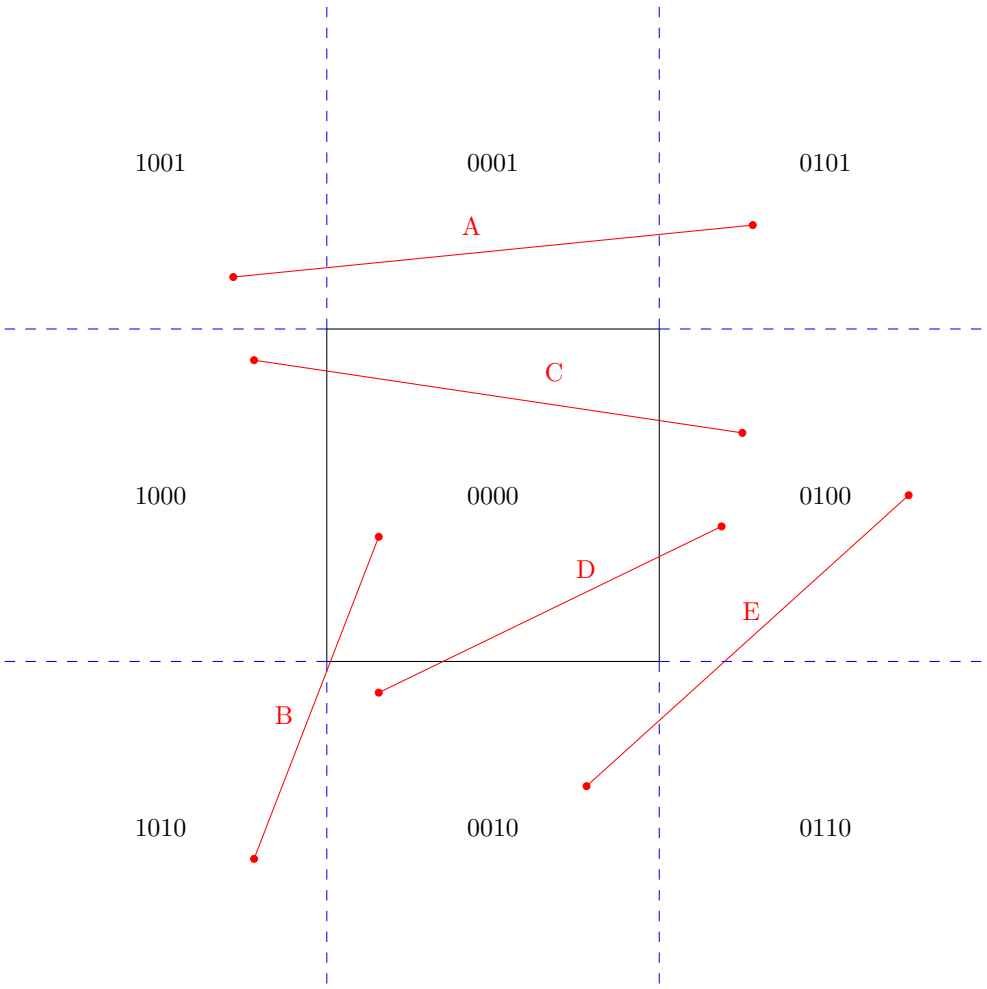


Fig. 2 Example cases of cell element intersection determination. Case A is thrown out by the ADT. Cases B and C are trivially marked as intersecting elements. Cases D and E must perform geometric predicates to determine intersection.

trivially accepted by noting that one of its out-codes is zero,

$$P_{B,1} = 0 \vee P_{B,2} = 0,$$

where $\vee$ is the or operator. Element C can be trivially accepted by noting that its two points lie between the y values of the cell boundaries or conversely the x values of the cell boundaries,

$$(P_{C,1} \mid P_{C,2} = 3) \vee (P_{C,1} \mid P_{C,2} = 12),$$

where $\mid$ is the bitwise or operator. If none of these cases are met, then geometric predicates for left/right determination must be used to determine intersection. Elements D and E provide an example showing elements with point out-codes, 0010 and 0100, that cannot determine element intersection. VACC makes use of the adaptive precision floating point library developed by Shewchuk in [14] for computing geometric predicates.

Once each cell has identified its intersected elements, all the cells with no intersection must be labeled as interior or exterior to the geometry. A painting algorithm marks regions in the domain as interior or exterior [4]. The painting algorithm uses a ray trace to determine if a single cell is interior or exterior and then 'paints' that result onto the region containing that cell and bounded by the geometry and domain boundaries. This allows inside outside determination to be performed without a ray-trace operation in every cell. A single ray trace acts as a seed for a breadth first search. Virtual perturbations are used for tie breaking geometric predicates when necessary [15].

After removing interior cells, the geometry intersection procedure computes new faces and inserts them into the cells. Figure 1 shows this as the fifth step. To perform a single intersection, a cell identifies a 'string' of elements that starts and ends outside the cell bounds with all intermediate points lying inside the cell bounds. A string consists of an arbitrary number of sequential geometry elements. After finding a string of elements, the entrance and exit points of the string are computed. These are the locations where the string first enters and last exits the cell being intersected. In Figure 1, these are the green circles. A new face is drawn between the entrance and exit points and labeled with an appropriate boundary condition. The faces that contain the entrance and exit points are split and then certain faces are flipped to maintain counterclockwise ordering on the compute volumes. The face data structure uses an edge to edge connectivity that allows rapid face traversal during this process. The convention for flipping faces is to take the ending half of the face split by the exit point and reversing faces around a counterclockwise loop until the entrance point is reached. Using this process, multiple strings can be intersected with a single cell as long as they do not intersect each other. After all intersections are complete, the remaining counterclockwise loops will mark compute volumes. VACC supports challenging situations so that a single cell can contain any non-negative number of counter clockwise loops.

This mesh generation procedure naturally supports adaptive mesh refinement (AMR). VACC supports AMR and can perform adaptations with solution injection based on geometric and flow criteria. When a cell is split it is divided into four isotropic elements. A geometric cell with refinement level, r, has size,

$$\text{Cell Size} = 2^{-r}\left[x_{\max} - x_{\min}, y_{\max} - y_{\min}\right]. \tag{1}$$

Storing a cell in memory requires knowing the location of its bottom left corner and its refinement level. The location of the bottom left corner is defined by integer coordinates on a uniform Cartesian grid on the finest refinement level. This allows the bounds of the geometric cells to be known exactly and stored compactly. The initial mesh used in the flow solution is often adapted according to geometric criteria to ensure the surface is properly resolved.

III. Inviscid Fluxes

The flow solver is a two dimensional second-order finite-volume method. VACC supports various explicit Runge-Kutta time stepping schemes of the form presented by Van Leer, Tai, and Powell [16]. A parallel scaling study is given in Sec. V along with a closer look at our time marching techniques.

A. Gradient Reconstruction

For simplicity, the flow solver uses a fully unstructured face centered approach. Gradient reconstruction is performed using a least-squares reconstruction technique [17]. Figure 3 presents the least-squares reconstruction. The least-squares system constructed from Figure 3 is,

$$\begin{bmatrix} r_{i,1,x} & r_{i,1,y} \\ r_{i,2,x} & r_{i,2,y} \\ r_{i,3,x} & r_{i,3,y} \\ r_{i,4,x} & r_{i,4,y} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \begin{bmatrix} u_1 - u_i \\ u_2 - u_i \\ u_3 - u_i \\ u_4 - u_i \end{bmatrix}, \tag{2}$$

where i represents the cell on which we are computing the gradient, u_i represents the state at cell i, and $r_{i,1,x}$ represents the x distance from cell i's centroid to cell 1's centroid. There is one equation in the least-squares system for each neighboring cell. This approach can then be easily extended to any number of neighbors greater than two by adding more equations. This allows cut-cells, volume cells and interface cells (cells that have neighbors of different refinement levels) to be treated the same way.

After constructing gradients, a limiter is applied to ensure that solutions remain total variation diminishing (TVD). The inspiration for this limiter is provided by work of Berger et al. [18, 19]. A linear programming (LP) problem is solved in each cell to ensure that it meets TVD and positivity conditions at each face. The objective of each LP problem is to maximize the value of the limiter,

$$\vec{\phi} = \phi_x \vec{x} + \phi_y \vec{y}, \tag{3}$$

subject to some TVD and positivity constraints. The value of ϕ_x and ϕ_y lies within $[0, 1]$. We split the value of the limiter into each coordinate direction to avoid unnecessary numerical diffusion introduced by scalar limiters [20]. Each face of a control volume contributes a constraint to its LP problem.

The faces that connect two control volumes enforce TVD constraints. Any compact, two point, limiter can be used to formulate the constraints. The two most common two point limiters are Barth-Jepersen and minmod [21]. In one

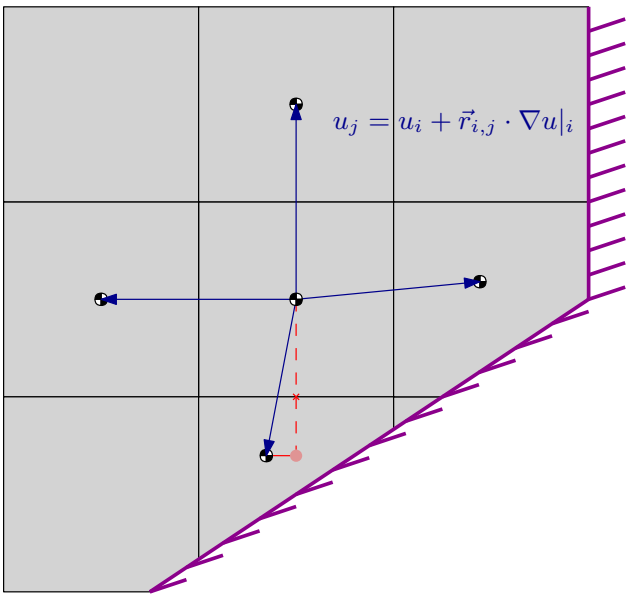


Fig. 3 The least-squares formulation for gradient reconstruction. The resulting system is given in Equation 2.

dimension, Barth-Jepersen (BJ) projects the state at a centroid to a neighboring face and then requires the projected value to be between the two neighboring centroid values. In two dimensions, this approach for the BJ limiter may limit a linear linear function if an isoline passes through two cell centroids, but not the connecting face centroid. Projecting the value at either state centroid to the face centroid will be interpreted as a peak or valley that needs to be flattened. To avoid unnecessarily limiting these linear solutions, BJ projects the state at a cell centroid to a face centroid and then enforces that it is within the minimum and maximum values over the original cell centroid value and all neighboring cells' centroid values. In one dimension, minmod ensures the state projected from a cell centroid to a neighboring cell centroid is between the neighboring cell centroid value and the original centroid value. This extends to two dimensions without the additional complication in BJ. For full Cartesian cells, the x and y directions are decoupled and the LP can be replaced with two 1D limiting problems. The horizontal faces are used to find ϕ_x and the vertical faces are used to find ϕ_y. At refinement level interfaces, there is a partial decoupling of the LP. The direction that is not a refinement level interface must be limited first. The limited gradient in the non-interface direction can then be used to recenter cell centroid values to apply the 1D limiter to each face in the other direction. An interface cell that changes refinement levels in both directions has no decoupling of the LP.

VACC has a directional BJ, scalar BJ and directional minmod limiters. The scalar limiter is easier to compute, but more diffusive since it does not solve an LP in each cell. The difference between the directional minmod and BJ limiters is the TVD constraints between cells. For BJ, the constraints on a cell given a face between a neighboring face is given as,

$$\begin{aligned}(\vec{\phi}_i \odot \nabla u) \cdot \vec{\Delta}_{i,f} + u_i &\geq \min_{\substack{k \in [i,j] \\ j \in N_i}} u_k, \\ (\vec{\phi}_i \odot \nabla u) \cdot \vec{\Delta}_{i,f} + u_i &\leq \max_{\substack{k \in [i,j] \\ j \in N_i}} u_k,\end{aligned} \tag{4}$$

where $\odot$ denotes element wise multiplication, $\vec{\Delta}_{i,j}$ denotes the vector from cell i centroid to face centroid f, and j provides an index over neighboring cells. The minmod constraints are more straightforward,

$$\begin{aligned}(\vec{\phi}_i \odot \nabla u) \cdot \vec{\Delta}_{i,j} + u_i &\geq \min_{k \in [i,j]} u_k, \\ (\vec{\phi}_i \odot \nabla u) \cdot \vec{\Delta}_{i,j} + u_i &\leq \max_{k \in [i,j]} u_k,\end{aligned} \tag{5}$$

with the state projected to neighboring centroids and compared directly.

Both directional limiters introduce positivity constraints at boundary faces. All free-stream boundaries are limited with ghost control volumes as described previously. The wall boundaries enforce positivity constraints. Since wall boundary faces are not coordinate aligned, the positivity constraints are enforced using scalar limiting,

$$(\vec{\phi}_i \odot \nabla u) \cdot \vec{\Delta}_{i,f} + u \geq \delta, \tag{6}$$

where $\vec{\Delta}_{i,f}$ is the distance from a cell centroid to a face centroid and δ is some small non-negative value. This constraint is only applied to pressure and density, the velocities are not constrained by positivity at boundaries. δ is used rather than zero to prevent the non-physical zero pressure or density. For density, δ is a fraction of the free-stream density,

$$\delta_\rho = 10^{-6}\rho_\infty. \tag{7}$$

For pressure, δ is a computed from the free-stream speed of sound and density in the cell,

$$\delta_p = 10^{-6}\frac{c_\infty^2 \rho}{\gamma}. \tag{8}$$

The LP is solved using the simplex method discussed by May and Berger [19]. This ensures that all projections remain physical and TVD. The scalar limiter is solved by looping over all faces in the mesh and decreasing the gradient in a cell if necessary. The scalar limiter applies the same positivity constraint as the directional limiters on boundary faces.

B. Second-Order Verification

The gradient reconstruction's second-order accuracy was verified using a supersonic vortex case [20, 22]. This case has an exact solution,

$$\begin{gathered} \rho = \rho_0 \left[1 + \frac{\gamma - 1}{2} M_0 \left(1 - \left[\frac{r_0}{r}\right]^2\right)\right]^{\frac{1}{\gamma - 1}} \\ u = a_0 M_0 \left(\frac{r_0}{r}\right) \sin\theta, \quad v = -a_0 M_0 \left(\frac{r_0}{r}\right) \cos\theta, \quad p = \frac{p_0}{\rho_0^\gamma}\rho^\gamma, \end{gathered} \tag{9}$$

where $[]_0$ indicates quantities at the inner radius. Figure 4 presents the flow domain for the coarsest mesh and the exact density solution. To verify the order of accuracy of VACC two types of tests were performed (Figure 5). The order of accuracy of each case is presented as a linear least-squares fit on the final three points of a given line.

The first test is the truncation error test. The truncation error test initializes the exact solution to every cell in the mesh. Then the residual is evaluated and the error is given as the sum of the absolute value of the residuals in every cell. This test was performed without a limiter and for the two BJ limiters. Without the limiter, the full cells achieve an order of 1.93, very close to the expected 2. The cut cells and subsequently all of the cells together experience an uptick in the residual at the finest mesh due to skewed stencils in small cells. Here, the order of accuracy presented for the cut cells and full cells are polluted by this last mesh. The limited solutions portray similar trends to the unlimited case, but at a worse order.

The second type of test is the fully converged test. In this test, a flow solution is converged and then the average L_1 error in the density at cell centroids is recorded. This test was also performed without a limiter and for the two BJ limiters. Again, the unlimited case gets close to the expected order of accuracy of 2, while the limited solution is slightly worse. The uptick visible on the final mesh in the truncation error test disappears when the solution is fully converged. The scalar limiter has an order of accuracy around 1.9, while the directional limiter retains an accuracy of 2 across all cells. It is also important to note that the directional limiter matches the magnitude of the error from the no-limiter case. The directional limiter does as well as the no-limiter case.

IV. Viscous Fluxes

The addition of viscous fluxes requires reconstructing gradients at face centroids as well as cell centroids. The two situations where this must be addressed are interior faces between control volumes and no-slip boundary faces. With the inviscid flux framework, the interior faces are easily addressed. The gradient's face-parallel component is computed by averaging the two gradients at the cell centroids. The gradient's face-normal component is computed by re-centering

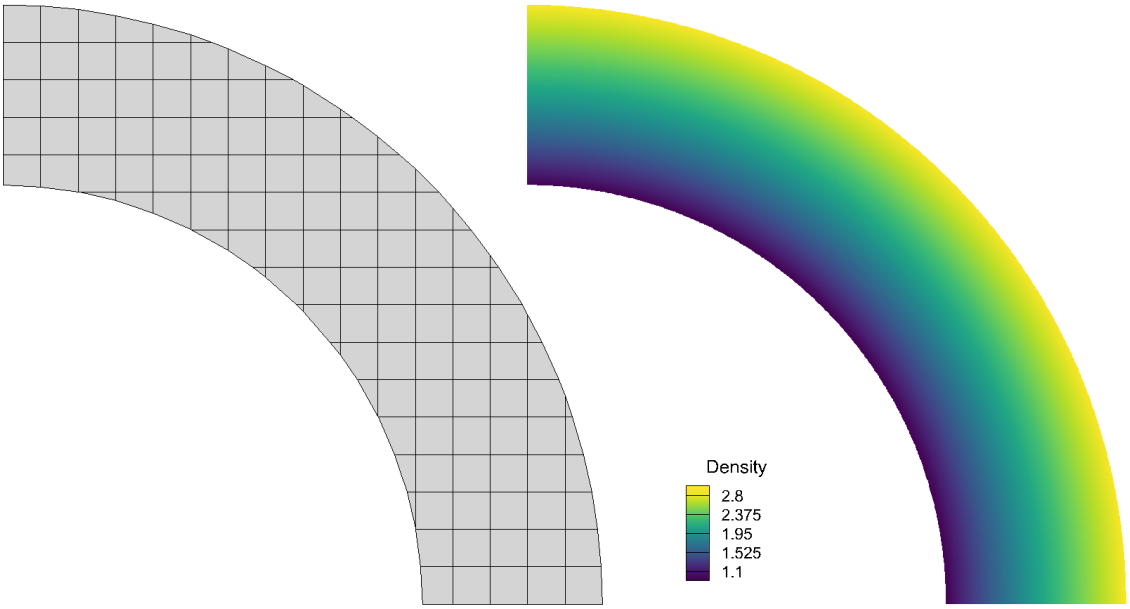


Fig. 4 The domain for the supersonic vortex verification. The density profile shown is the exact solution.

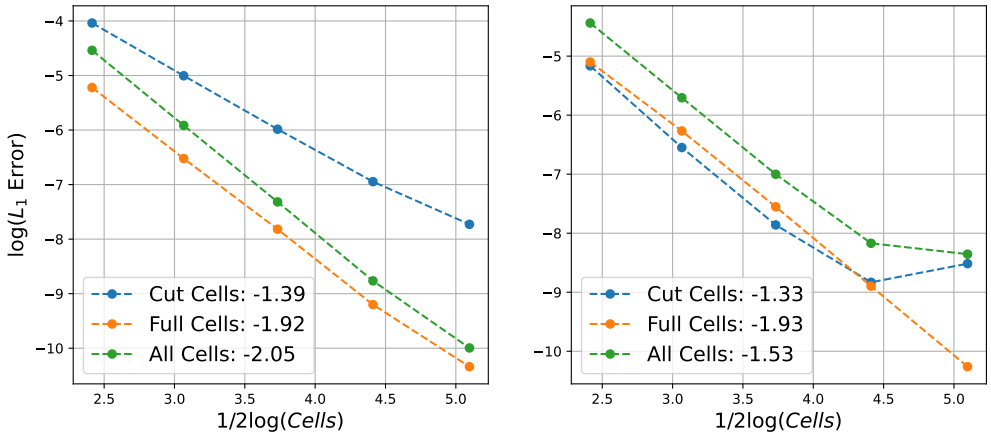
cell centroid states as seen in Figure 6 and then computing a gradient at the face from a central difference. Averaging the face normal gradients like the face parallel gradients would result in an unstable 5-point stencil [23].

Gradients at the wall cannot be approximated by using the cell-centroid gradient. At the adiabatic no-slip boundaries only velocity gradients are needed. Figure 7 presents a diagram to motivate the discussion of computing velocity gradients at the wall. The initial intuition for computing a gradient at the wall would be to use the gradient at the cell centroid of the cut-cell. However, incompressible flat-plate boundary layer theory indicates a quadratic tangential velocity profile [24]. Cut-cells have their cell centroid at an arbitrary distance from the wall, making it difficult to predict the gradient at the wall without outside information. It is also important to note that with varying sizes of cut-cells, the difference in gradient applied in each neighboring cut-cell may be large. This would result in a skin-friction that appears jagged. To get around this, we follow the work of Berger and Aftosmis [9].

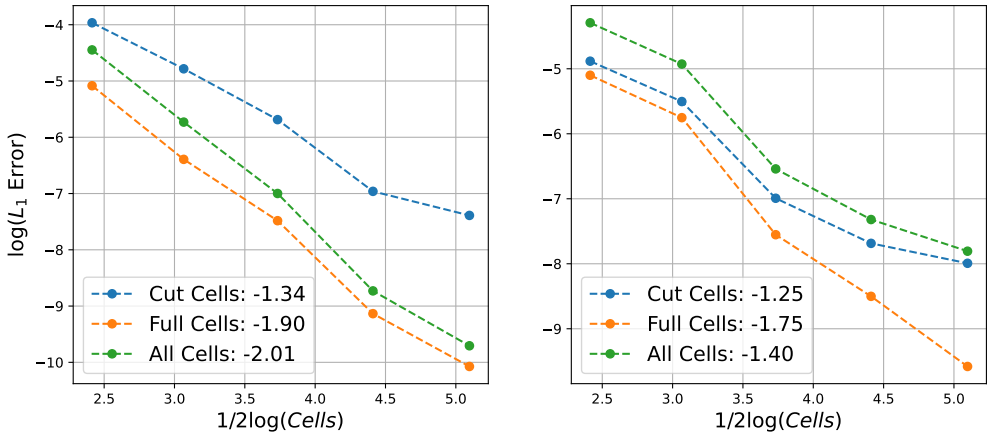
To compute the velocity gradients at the wall, a quadratic reconstruction of the tangential velocity is performed in each cut cell. Each cut cell finds its neighboring cell in the wall-normal direction through its own cell centroid. The velocity vectors and the velocity gradients of both cells are rotated into the wall-normal and wall-tangential coordinate frame. Then tangential velocities of the cut cell and the next level cell are used to construct a quadratic tangential velocity profile. The second level cell could recenter its gradient in the wall-tangential direction, but for simplicity we assume that this gradient is much less than the wall-normal gradient and therefore do not recenter it. After the tangential velocity profile is constructed, the gradient at the wall can be computed from the quadratic and used for evaluating the flux at the wall. Additionally, the tangential velocity's derivative in the wall-normal direction is replaced with the derivative computed from this quadratic. This updated cut-cell gradient is rotated back into the x/y coordinate frame and used to compute the fluxes on the other faces.

A. Flat-Plate Boundary Layer

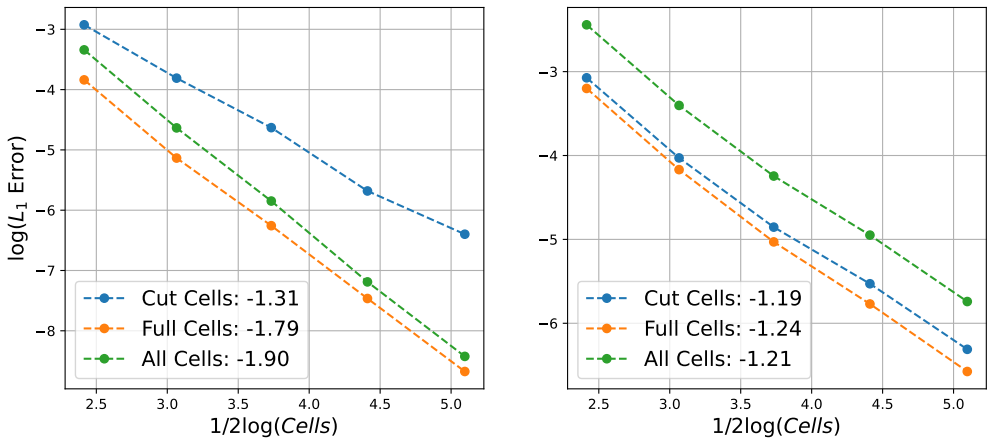
To test our implementation of the viscous fluxes, a Blasius flat plate was simulated. Two cases were considered: a coordinate-aligned flat plate and a flat plate rotated 15°. The rotated case introduces cut-cells with areas varying over eight orders of magnitude. Figure 8 presents the full computational domain and a close up look at the mesh in the boundary layer for both cases. In both cases, the reference length is taken to be 1 and $Re = 5000$. The computational domain is forty lengths in both x and y directions. The 0° case does not have any cut-cells in it and uses the bottom boundary as the flat plate. The 15° case cuts the flat plate out of the square domain at a 15° angle. In both cases, the plate when $x < 0$ is set to an inviscid wall boundary condition. For $x > 0$ the plate is set to an adiabatic no-slip boundary condition. All other boundary conditions were set to free stream, with the boundaries far enough away to have a negligible effect on the solution.



(a) No limiter



(b) Barth-Jepersen LP



(c) Barth-Jepersen Scalar

Fig. 5 The order of accuracy tests for the supersonic vortex case. The legends present the computed order of accuracy from a least-squares fit to the data.

To compare the effect of varying numbers of cells in the boundary layer, velocity profiles were extracted from the mesh at different lengths down the flat plate. Table 1 identifies the stations where velocity profiles were taken. Figure 9

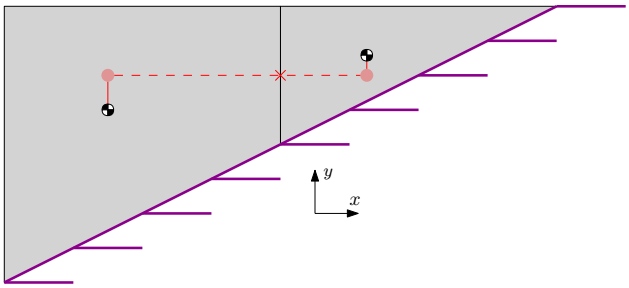


Fig. 6 The re-centering approach taken to compute gradients at face centroids for cut-faces and interfaces.

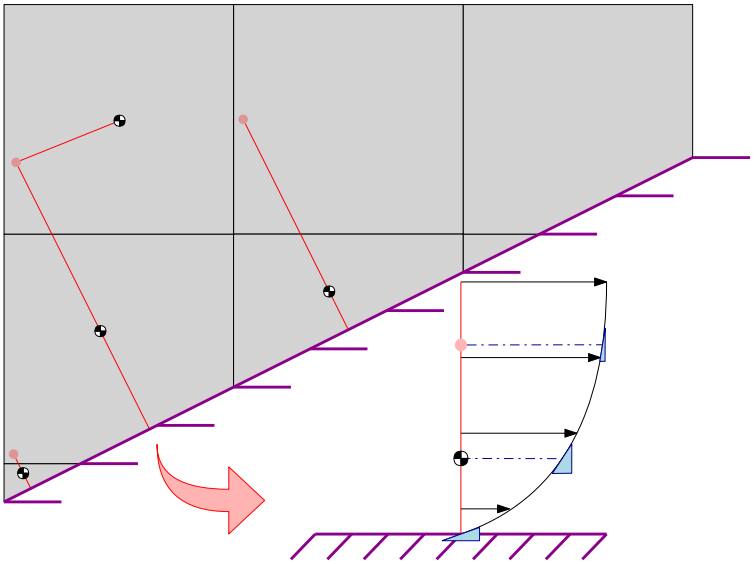


Fig. 7 A diagram to explain the computation of velocity gradients near a no-slip wall. The gradient in the cut cell cannot be used at the wall because it may be a poor approximation. These gradients can also vary as we traverse the wall, since cut cell centroids are at varying distances from the wall.

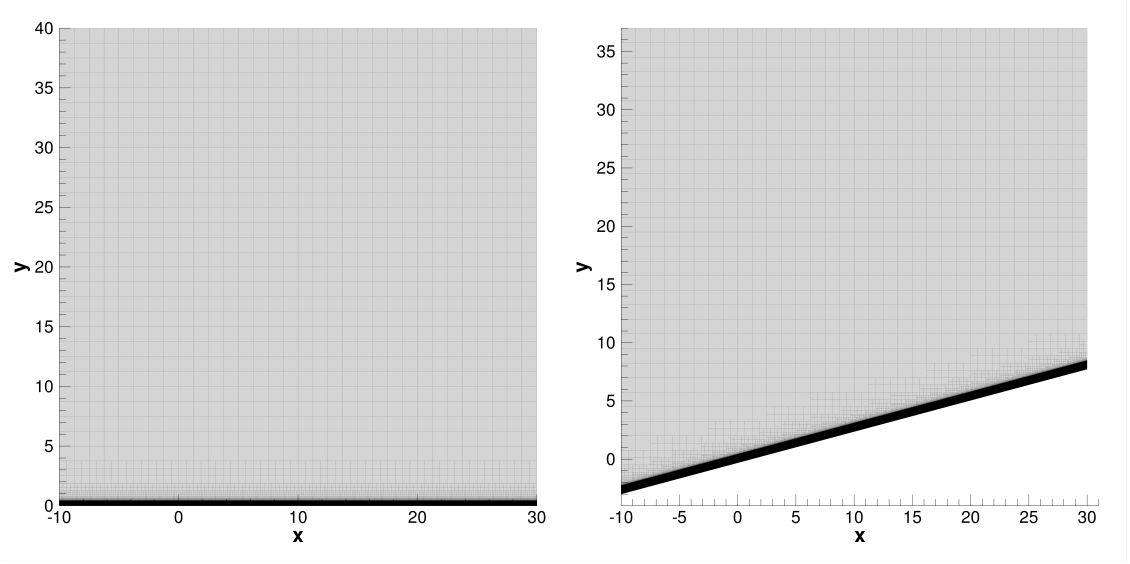
Table 1 Velocity profile stations for the flat plate cases. For the rotated case, x is taken as the distance tangential to the flat plate starting at x = 0. The number of cells in the boundary changed slightly when rotating the grid. The exact numbers of cells can be found in Figure 9.

X Location	Re_x	Cells in Boundary
1	5,000	≈ 15
2	10,000	≈ 20
10	50,000	≈ 45

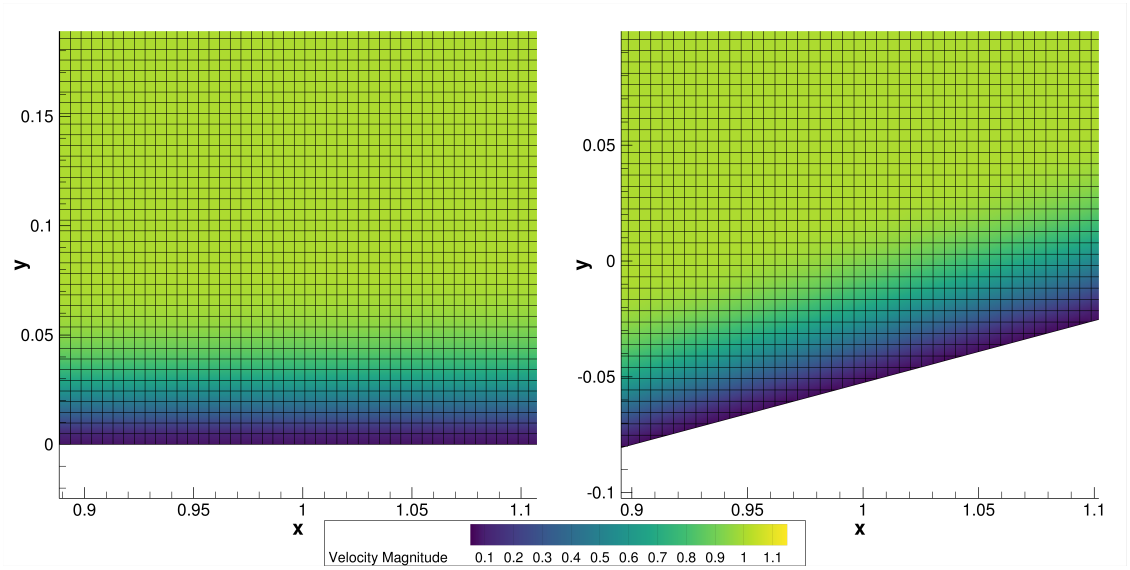
presents the velocity profiles at these three stations compared to the Blasius solution. The legends in Figure 9 identify the exact number of cells in the boundary layer at the given stations. The boundary layer is defined as the cells for which the velocity magnitude is 99% of the free stream value,

$$\|\vec{v}\| \leq 0.99U.$$

The wall-aligned velocities match the Blasius solution well, even in the 15° rotation case with cut cells. The wall-normal velocity overshoots by a small margin, but this shrinks as the mesh is refined. Figure 10 shows the skin friction along the length of the plate. A point is taken from every cut cell. After $Re_x = 100$, both solutions match the Blasius solution well.



(a) The full domain for the rotated and unrotated Blasius cases.

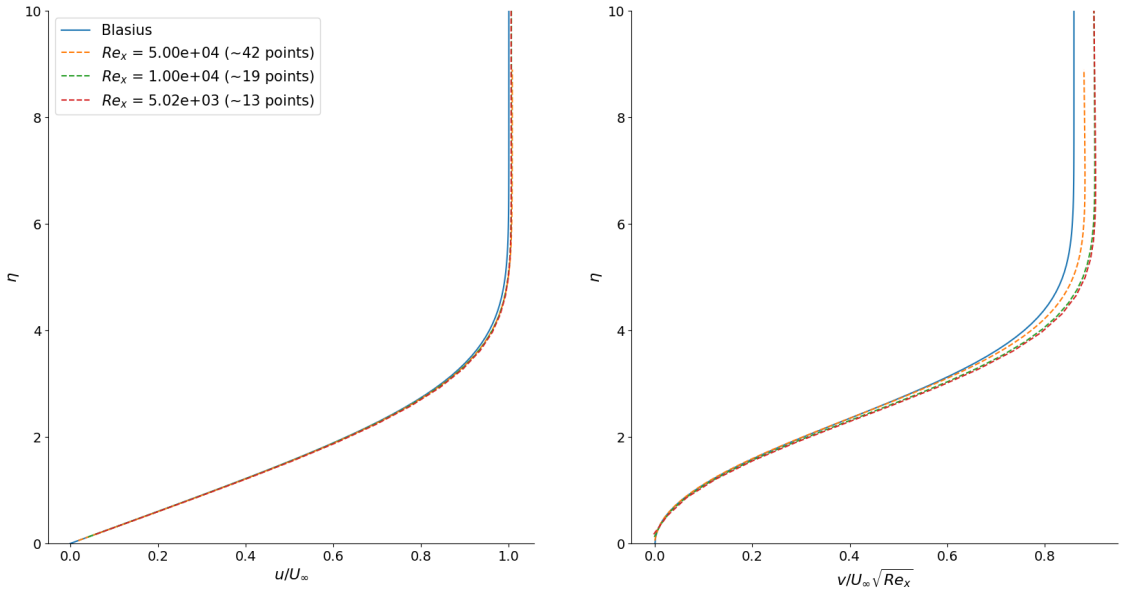


(b) A close-up look at the mesh in relation to the boundary layer at the $Re_x = 5000$ station.

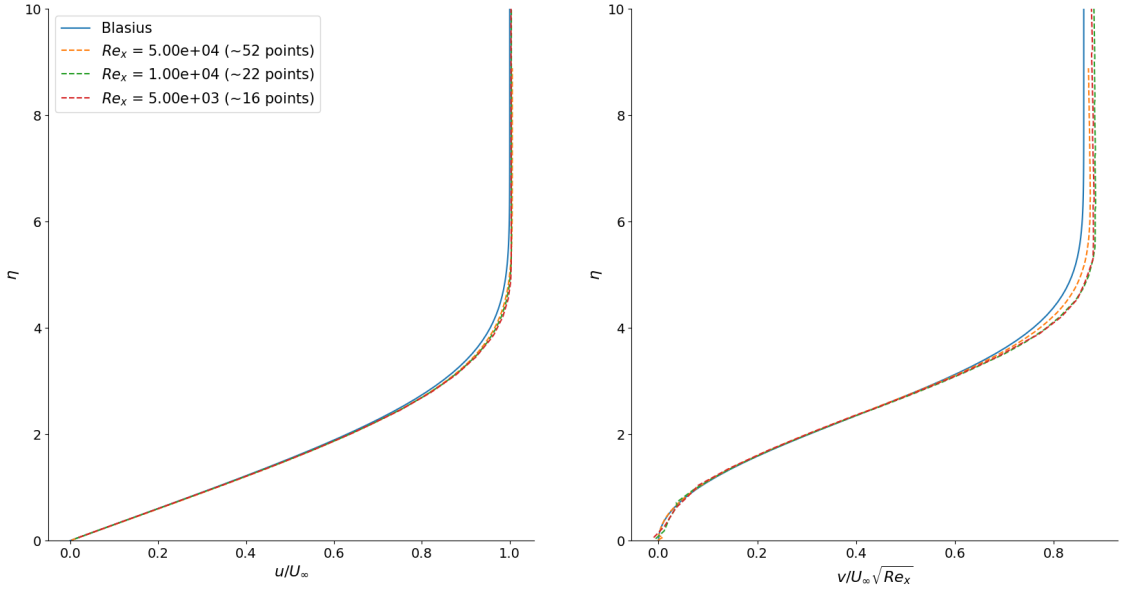
Fig. 8 The computational domain for two flat plate boundary layer cases. The velocity profile stations are at $x = [1, 2, 10]$.

B. NACA 0012

A NACA 0012 airfoil case was run at Mach 0.5, Reynolds number 5,000 and zero angle of attack. This introduces the additional challenge of wall curvature. The grid for this case was created by starting with a level six uniform mesh 100 by 100 chord lengths. The airfoil was placed in the center and all cut-cells were tagged for refinement ten times. During each refinement, ten breadth-first sweeps were made to provide a smooth transition between refinement levels. The final grid had about 54,000 cells. Figure 11 shows a contour of the velocity magnitude. There are two close ups for viewing the mesh near the leading-edge and the separation bubble at the trailing edge of the airfoil. Figure 12 indicates a smooth skin friction. The high curvature leading-edge and lack of sufficient grid points results in a minutely jagged skin friction. This recovers to a smooth skin friction distribution once $x/c = 0.1$ is reached. Sufficient resolution would reduce these oscillations. However, we were not able to converge a steady solution with additional grid points using our explicit time-marching scheme. Too many grid points, especially in the wake region, yield resolution sufficient to



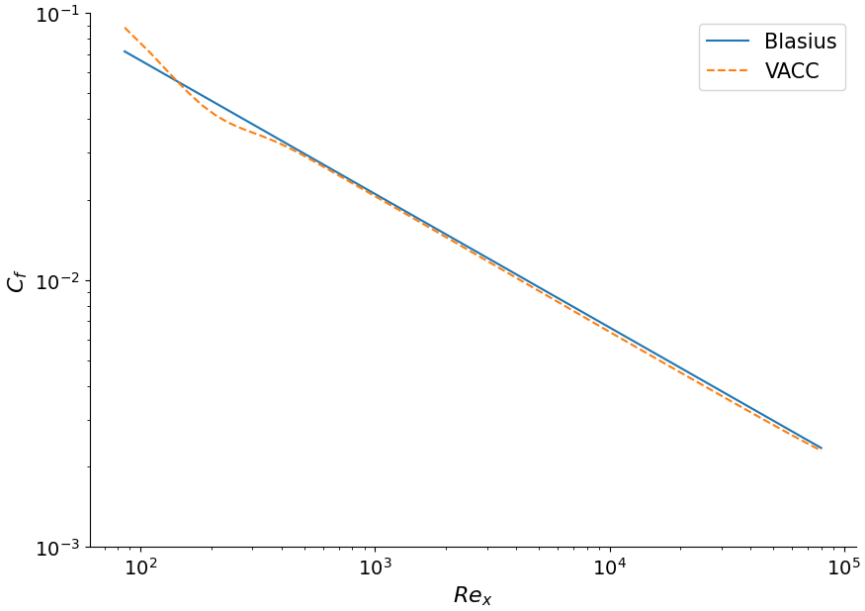
(a) The velocity profiles for the 0° rotated case.



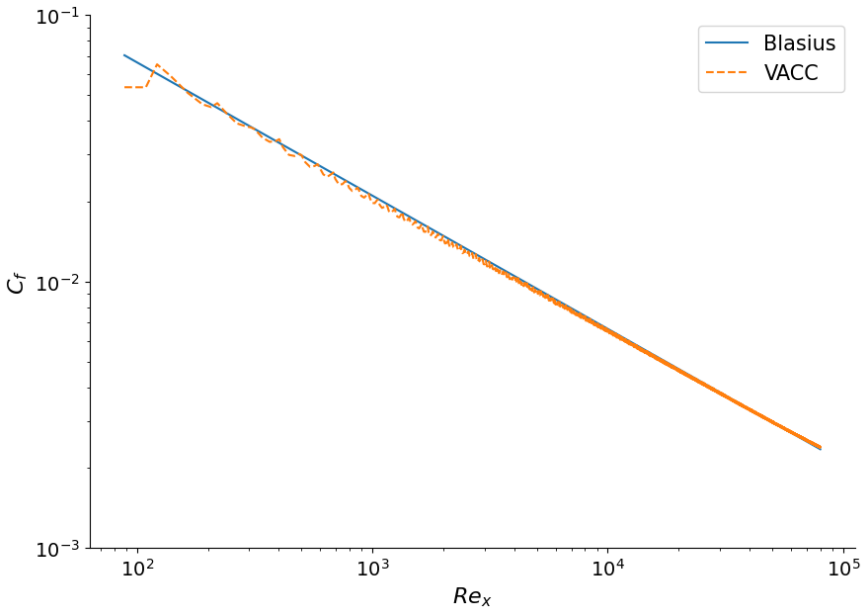
(b) The velocity profiles for the 15° rotated case.

Fig. 9 The velocity profiles for the 0° and 15° rotated cases. Each plot includes the Blasius solution for comparison. The legend indicates the number of cells that were in the boundary layer, defined as cells within the 99% thickness.

resolve the unsteady solution and vortex shedding. The separation point for the steady solution was at 0.841, which is within 5% of the mesh-converged values of the separation point reported by Swanson and Langer [25].



(a) The skin friction along the flat plate for the 0° rotated flat plate.



(b) The skin friction along the flat plate for the 15° rotated flat plate.

Fig. 10 The skin frictions for the 0° and 15° rotated cases. Each plot includes the Blasius solution for comparison. After getting past the stagnation point, the skin frictions match Blasius solution well.

V. Iteration Strategies

The base steady-state solver is a forward Euler time marching scheme,

$$u_i^{n+1} = u_i^n - \frac{\Delta t^n}{A_i} \sum_{e=1}^{N_n} \hat{F}_{i,e} \Delta l_{i,e}, \tag{10}$$

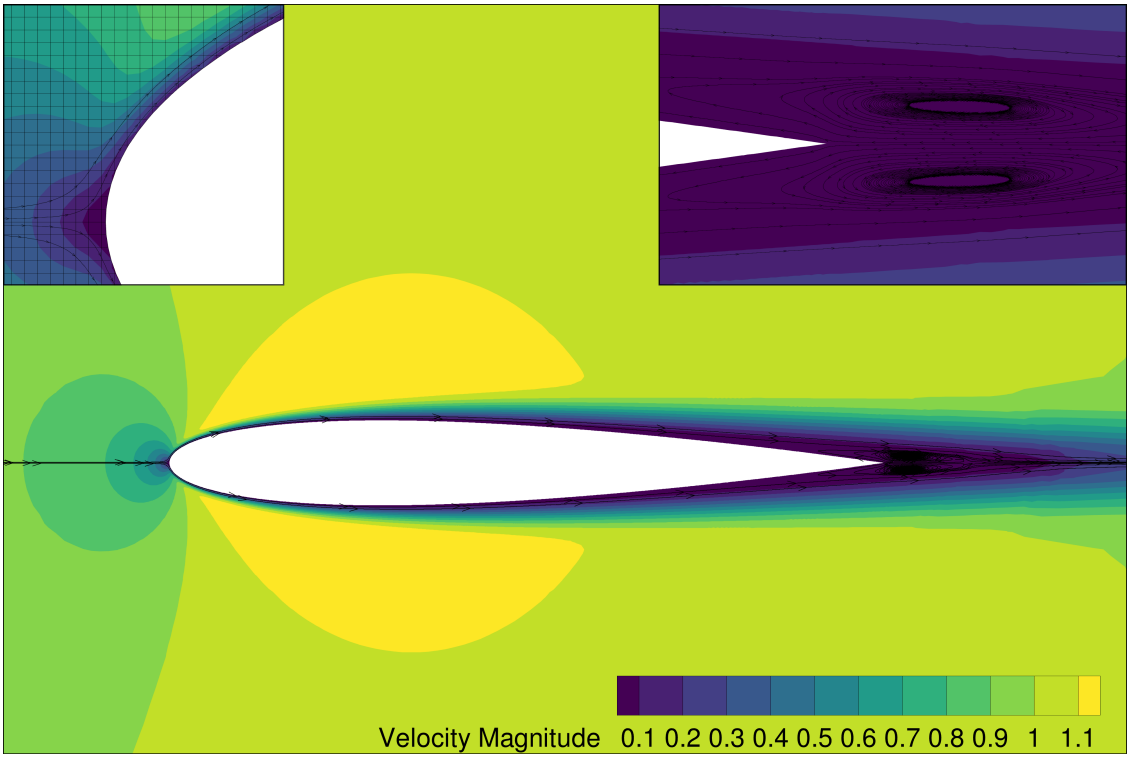


Fig. 11 The velocity magnitude about a NACA 0012 airfoil run at Mach 0.5, Reynolds number 5,000 and zero angle of attack.

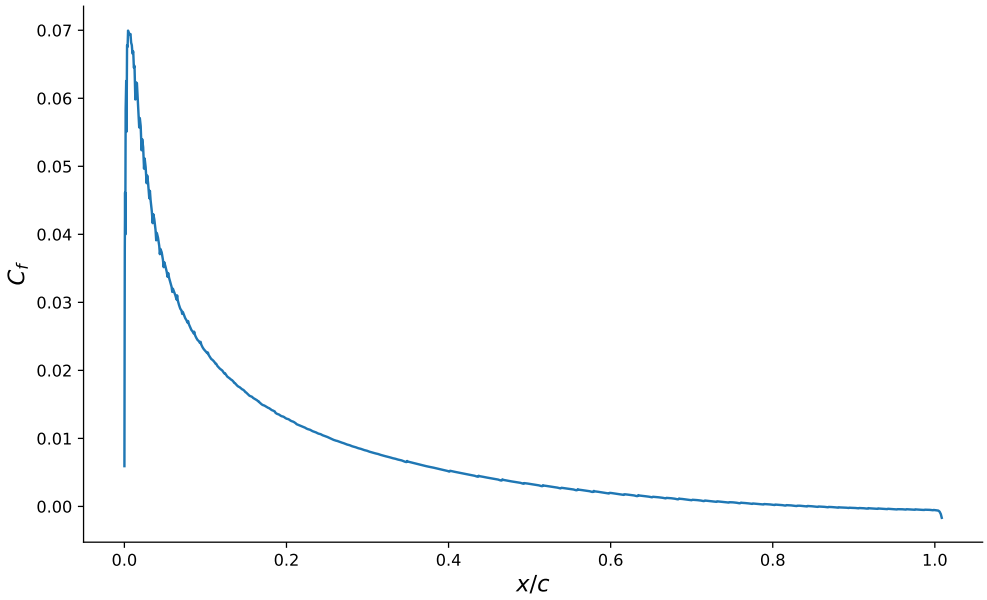


Fig. 12 The skin friction for the NACA 0012 airfoil case. The high curvature and poor spatial resolution results in some roughness, especially around the leading-edge.

where $\hat{F}_{i,e}$ is the Roe flux evaluated at the face between cells i and e, A_i is the area of cell i, and N_n is the number of cells neighboring cell i. The time step is computed locally in each cell according to,

$$\Delta t_i^n = \frac{A_i \text{CFL}}{\sum_{e=1}^{N_n} |s_{i,e}| \Delta l_{i,e} + 2\nu_i}, \tag{11}$$

where $s_{i,e}$ is the maximum wave speed computed by the Roe flux at the face between cell i and e, and ν is the maximum of the viscosity and thermal diffusivity,

$$\nu = \max(\nu, k/(\rho c_v)). \tag{12}$$

The first term in the denominator of the time step corresponds to the inviscid fluxes and the second term corresponds to the viscous fluxes [26]. The time step in cut-cells is halved to ensure robustness with the poor stencils close to the boundary. To accelerate the convergence of the solver to steady state, the following techniques were explored: Runge-Kutta (RK) multi-stage time stepping, implicit residual smoothing, p-multigrid, and MPI parallelization.

A. Runge-Kutta Time Stepping

To ensure that the time stepping method would be stable when including viscous terms the forward Euler time marching scheme was extended to include RK schemes in the format present by Van Leer [16],

$$\begin{aligned} u_i^{(0)} &= u_i^n, \\ &\vdots \\ u_i^{(k)} &= u_i^{(0)} + \alpha_k \frac{\Delta t_i^n}{A_i} R_i(u^{(k-1)}), \\ &\vdots \\ u_i^{(n+1)} &= u_i^{(m)}, \end{aligned} \tag{13}$$

where m is the number of RK stages, R_i is the flux residual computed in cell i from state $u^{(k-1)}$, and α_k is the RK coefficient for that stage. The RK coefficient for the last stage, α_m, is always 1 since it spans the full time step. Fully evaluating the residuals at each RK stage is not always necessary, although it is always more robust. To speed up convergence, gradients and fluxes may be updated only at certain intermediate stages. Table 2 presents the effect of different RK schemes on the convergence speed for the coordinate-aligned Blasius case from Sec. IV.A with no limiters. The RK coefficients and CFL's chosen for each scheme were the optimal smoothing coefficients for a second-order

Table 2 Runtime comparison for different RK schemes on the coordinate aligned case from Sec. IV.A. The RK stage coefficients were the optimal multi-stage coefficients for a second-order scheme in [16]. Gradients were only evaluated at the beginning of the time step, all intermediate stages did not recompute gradients. All wall times were recorded on four full Skylake nodes or 160 processors on NASA's Electra supercomputer.

RK Stages	CFL	Wall Time [s]	Total Iterations
2	0.4693	15,047	4,478,400
3	0.6936	14,319	2,994,830
4	0.9214	13,655	2,246,787
5	1.1508	13,596	1,797,455

scheme, given in [16]. It is reasonable to change the CFL number for these comparisons because as the number of stages increases, the stability region of the scheme also increases. For this comparison, gradients were only evaluated before the first stage. Every stage still replaced the tangential velocity gradient at each stage, but the least-squares was only computed at the beginning of each time step. The wall times reported were measured on four Skylake nodes (160 processors) on NASA's Electra supercomputer. As the number of stages increased, the time to convergence decreased.

B. Implicit Residual Smoothing

Implicit residual smoothing can be used to increase the CFL limit and speed up convergence [27, 28]. For an unstructured grid, central implicit residual smoothing (CIRS) is implemented through a Laplacian operator. The smoothed residual in cell i is given as,

$$\vec{R}_i^* + \sum_{e=1}^{N_n} \epsilon \left(\vec{R}_i^* - \vec{R}_e^* \right) = \vec{R}_i, \tag{14}$$

where R^* indicates smoothed quantities and ϵ indicates the smoothing factor. Based on the work done by Lassaline [27] we found $\epsilon = 0.175$ to be a good value that allows the CFL number to be doubled. Equation 14 must be rearranged into an iterative form before implementation,

$$\vec{R}_i^{(n+1)} = \frac{\vec{R}_i^{(0)} + \epsilon \sum_{e=1}^{N_n} \vec{R}_e^{(n)}}{1 + N_n \epsilon}. \tag{15}$$

This system is diagonally dominant and typically convergences in only two or three iterations. VACC implements this smoothing operation as two passes over the residuals every time they are evaluated. For the coordinate-aligned flat plate case with a five-stage scheme, the wall time reduces from 13,596 seconds to 8,465 seconds, a 37% decrease.

C. Two Level Correction

VACC also includes the option for full approximation storage (FAS) p-multigrid. The derivation of this method for a second-order finite-volume code is trivial. Since the grid does not change between levels of the multigrid, the prolongation and restriction operators become the identity matrix. Therefore, iterating the state with the first-order method directly updates the second-order state. The only modification that needs to be made to the first-order method is the addition of a source term,

$$S = R_1(u^0) - R_2(u^0), \tag{16}$$

where R_1 is the first-order residual evaluation, R_2 is the second-order residual evaluation, and u^0 is the state vector before the first first-order step. There are only two easily available levels for p-multigrid on a second-order code: first and second-order. This makes the only available multigrid cycle a two-level correction.

During the first-order residual evaluations, all gradients are frozen. The least-squares algorithm is not run and the tangential velocity gradients in cut-cells are not updated when computing the gradient at the wall. This does not allow the first-order method to converge the viscous fluxes at all. However, it accelerates the convergence of the inviscid fluxes.

Table 3 presents timings for various combinations of pre and post iterations for the two-grid correction. A five-stage

Table 3 Runtime comparison for different multigrid schemes on the coordinate aligned case from Sec. IV.A. The MG Scheme is defined as a-b, where a and b are the number of second-order and first-order iterations in a MG cycle, respectively. The same five-stage RK scheme from Table 2 was used, least-squares gradients were only computed once per time step. All wall times were recorded on four full Skylake nodes, 160 processors, on NASA's Electra supercomputer.

MG Scheme	Wall Time [s]	2^{nd} Order Iters.	1^{st} Order Iters.	MG Cycles
1-1	14,066	898,730	898,729	898,729
1-2	13,520	599,682	1,199,362	599,681
1-3	13,154	449,828	1,349,476	449,872
1-4	13,192	360,188	1,440,740	360,187
1-5	13,092	300,456	1,502,128	300,455
2-1	14,079	1,198,305	599,152	599,152
2-2	13,478	899,499	899,497	499,749
2-3	13,466	719,683	1,079,520	359,841
2-4	12,989	600,274	1,200,539	300,136
2-5	13,027	514,594	1,286,469	257,296

RK scheme with optimal stage coefficients and CFL was used to record these times. The RK scheme only computed the

least-squares gradients before the first stage. The data in Table 3 can be fit to a linear least-squares approximation to determine the cost of a second-order iteration, first-order iteration and a second-order residual evaluation with gradient computation. This results in approximate timings for the second and first-order iterations as 7.62 ms and 6.88 ms on 160 cores, respectively. The additional second-order residual evaluation, which is equivalent to the extra computational overhead for a multigrid cycle, is approximately 1.19 ms. The cost of a first-order residual evaluation is similar to a second-order evaluation because we are only avoiding a single gradient calculation at the beginning of the time step. Both the first and second-order residual evaluations still have to be evaluated five times, which is the bulk of a time step's cost.

Naturally, the next consideration is p-multigrid when all the gradients in the RK scheme are turned on. Converging the Blasius solution with a five-stage RK scheme with all gradients turned on results in a 160 processor wall time of 17,109 s. Applying a 2-3 p-multigrid cycle to this RK scheme reduces the 160 processor wall time to 15,219 s, an 11% speed up in convergence time. However, when running these same two cases with a five-stage RK scheme with only one gradient evaluation, the run time is reduced to 13,596 s without multigrid and 13,466 s with multigrid. These give an approximate 20% speedup compared to the full gradient evaluation scheme, but the difference between the two single gradient evaluation schemes is <1%.

When running difficult cases where additional robustness is needed, p-multigrid may be useful. The RK schemes and CFL condition implemented in VACC guarantee a physical solution when the gradients are evaluated and limited at each stage. However, they do slow the convergence time down significantly, so in general only one gradient evaluation may be used per time step. When a difficult case is having trouble remaining physical, it may be necessary to turn on additional gradients. In this case, when the second-order time step cost begins to increase, the benefits of p-multigrid will become evident.

D. MPI Parallelization

The grid is partitioned using a Morton order space-filling curve [29]. Figure 13 presents an example domain decomposition based on the space filling curve.

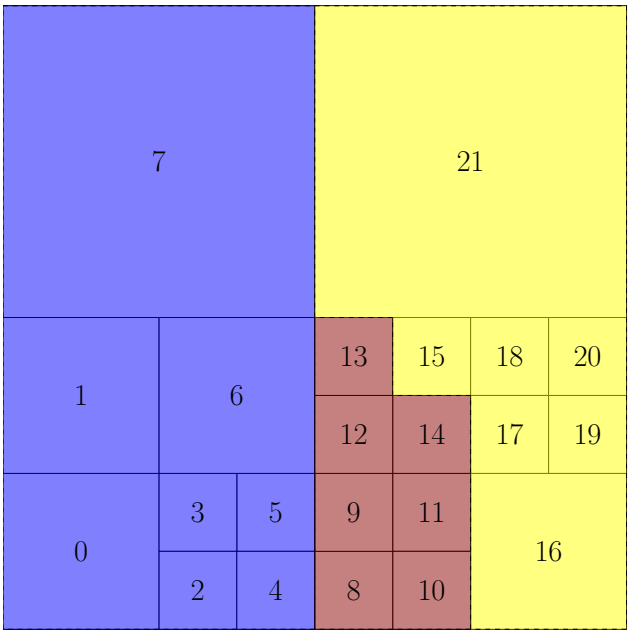


Fig. 13 An example of the space-filling curve used to partition the grid. Each processor receives a similar number of cells from a range on the space filling curve.

A strong scaling study was completed to verify the scalability of VACC (Figure 14). The two grids are 200 thousand cells and 729 thousand cells versions of the 15° Blasius case. This ensures that the parallel performance is investigated with viscous effects and no-slip boundaries. Both grid sizes drop off in efficiency early and then go super-linear. This is attributed to cache access patterns across different numbers of processors. When the efficiency drops below 1 a second

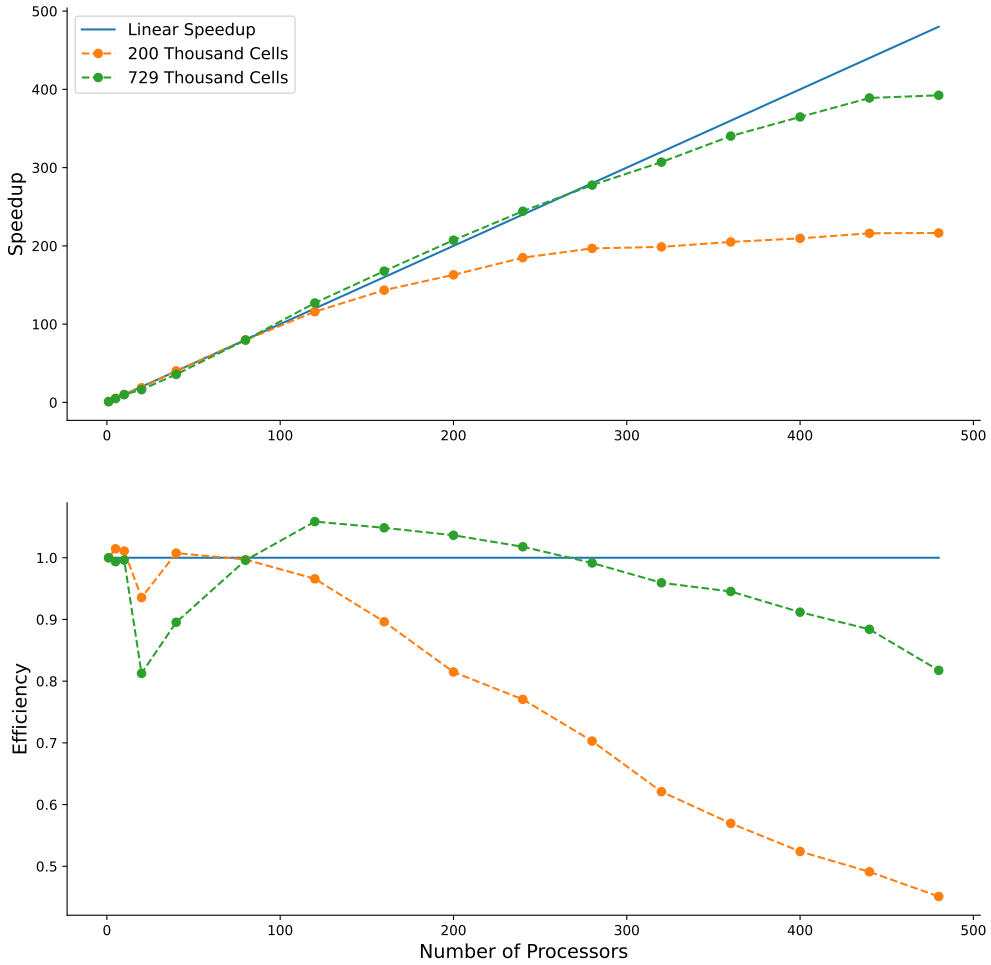


Fig. 14 Strong scaling study performed on two different size meshes. All times are recorded on Skylake nodes on NASA's Electra supercomputer. Each data point is taken as the median wall time over five samples of 2000 second-order iterations.

time, both cases are down to about 2,500 cells per processor.

VI. Conclusion

VACC is a second-order finite-volume code that provides a foundation for future research in viscous Cartesian cut cells. It employs least-squares gradient reconstruction with directional LP limiting in each cell to ensure TVD. Viscous fluxes in the domain are computed using central differences and gradient averaging. Viscous fluxes are computed at the wall using a quadratic velocity profile reconstruction. This reconstruction allows us to compute smooth skin frictions with varying cell size at the wall. VACC has demonstrated competency for solving the laminar Navier-Stokes on a variety of cases. Extending current inviscid methods to high Reynolds number flows will require combining the robustness of cut cell methods with the accuracy of boundary conforming methods. Future tasks include implementing an adjoint for adaptive mesh refinement and the investigation of the RANS equations.

Acknowledgments

We thank Marian Nemec, Mike Aftosmis, and the rest of the Cart3D team for insightful discussions on how to properly incorporate viscous terms into a Cartesian cut-cell framework. We would also like to thank Bernardo Pacini, Christian Jacobsen, and Jenna King for there help with the original implementation. Berardo also wrote the final ADT and painting algorithms. A. Kleb was supported by the NASA Office of STEM Engagement (OSTEM) and the National Science Foundation Graduate Research Fellowship Program under Grant No. (DGE 1841052). Computational resources were provided by the NASA High-End Computing Program through the NASA Advanced Supercomputing Division at Ames Research Center.

References

[1] Martins, J. R. R. A., "Aerodynamic Design Optimization: Challenges and Perspectives," *Computers & Fluids*, Vol. 239, 2022, p. 105391. https://doi.org/10.1016/j.compfluid.2022.105391.

[2] Seraj, S., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of a Supersonic Transport Considering Low-Speed Stability," *AIAA SciTech Forum*, 2022. https://doi.org/10.2514/6.2022-2177.

[3] Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," Tech. Rep. CR–2014-218178, NASA, March 2014. URL https://ntrs.nasa.gov/citations/20140003093.

[4] Aftosmis, M. J., "Lecture Notes in Solution Adaptive Cartesian Grid Methods for Aerodynamic Flows with Complex Geometries," *VKI Lecture Series*, 1997.

[5] Aftosmis, M. J., Berger, M. J., and Melton, J. E., "Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry," *AIAA Journal*, Vol. 36, No. 6, 1998.

[6] Coirier, W. J., and Powell, K. G., "Solution-adaptive Cartesian cell approach for viscous and inviscid flows," *AIAA Journal*, Vol. 34, No. 5, 1996, pp. 938–945.

[7] Zhao, H., Hu, P., Kamakoti, R., Dittakavi, N., Aftosmis, M., Marshall, D., Xue, L., Ni, K., and Mao, S., "Towards efficient viscous modeling based on Cartesian methods for automated flow simulation," *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010, p. 1472.

[8] Ye, T., Mittal, R., Udaykumar, H., and Shyy, W., "An accurate Cartesian grid method for viscous incompressible flows with complex immersed boundaries," *Journal of computational physics*, Vol. 156, No. 2, 1999, pp. 209–240.

[9] Berger, M., and Aftosmis, M., "Progress towards a Cartesian cut-cell method for viscous compressible flow," *50th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2012, p. 1301.

[10] Wissink, A., Katz, A., Chan, W., and Meakin, R., "Validation of the strand grid approach," *19th AIAA Computational Fluid Dynamics*, 2009, p. 3792.

[11] Capizzano, F., "Turbulent wall model for immersed boundary methods," *AIAA Journal*, Vol. 49, No. 11, 2011, pp. 2367–2381.

[12] Berger, M. J., and Aftosmis, M. J., "An ODE-based wall model for turbulent flow simulations," *AIAA Journal*, Vol. 56, No. 2, 2018, pp. 700–714.

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

[14] Richard Shewchuk, J., "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete & Computational Geometry*, Vol. 18, No. 3, 1997, pp. 305–363.

[15] Edelsbrunner, H., and Mücke, E. P., "Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms," *ACM Transactions on Graphics (TOG)*, Vol. 9, No. 1, 1990, pp. 66–104.

[16] Van Leer, B., Tai, C.-H., and Powell, K., "Design of optimally smoothing multi-stage schemes for the Euler equations," *9th Computational Fluid Dynamics Conference*, 1989. https://doi.org/10.2514/6.1989-1933.

[17] Ollivier-Gooch, C., and Van Altena, M., "A high-order-accurate unstructured mesh finite-volume scheme for the advection–diffusion equation," *Journal of Computational Physics*, Vol. 181, No. 2, 2002, pp. 729–752.

[18] Berger, M., Aftosmis, M. J., and Murman, S. M., "Analysis of Slope Limiters on Irregular Grids," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, 2005. https://doi.org/10.2514/6.2005-490.

[19] May, S., and Berger, M., "Two-Dimensional Slope Limiters for Finite Volume Schemes on Non-Coordinate-Aligned Meshes," *SIAM Journal on Scientific Computing*, Vol. 35, No. 5, 2013, pp. A2163–A2187. https://doi.org/10.1137/120875624, URL https://doi.org/10.1137/120875624.

[20] Aftosmis, M., Gaitonde, D., and Tavares, T. S., "Behavior of linear reconstruction techniques on unstructured meshes," *AIAA Journal*, Vol. 33, No. 11, 1995, pp. 2038–2049. https://doi.org/10.2514/3.12945.

[21] Barth, T., and Jepersen, D., "The design and application of upwind schemes on unstructured meshes," *27th Aerospace Sciences Meeting*, 1989. https://doi.org/10.2514/6.1989-366, URL https://arc.aiaa.org/doi/abs/10.2514/6.1989-366.

[22] Nemec, M., and Aftosmis, M. J., "Toward automatic verification of goal-oriented flow simulations," Tech. rep., August 2014.

[23] Steinthorsson, E., Modiano, D., Crutchfield, W., Bell, J., and Colella, P., *An adaptive semi-implicit scheme for simulations of unsteady viscous compressible flows*, 1995. https://doi.org/10.2514/6.1995-1727, URL https://arc.aiaa.org/doi/abs/10.2514/6.1995-1727.

[24] White, F. M., *Viscous Fluid Flow*, McGraw-Hill, New York, NY, 1991.

[25] Swanson, R. C., and Langer, S., "Comparison of NACA 0012 laminar flow solutions: structured and unstructured grid methods," Tech. rep., 2016.

[26] Hirsch, C., *Numerical Computation of Internal and External Flows*, Vol. 2, 1990.

[27] Lassaline, J. V., "A navier-stokes equation solver using agglomerated multigrid featuring directional coarsening and line-implicit smoothing," Ph.D. thesis, University of Toronto, 2003.

[28] Blazek, J., *Computational Fluid Dynamics: Principles and Applications*, third edition ed., Butterworth-Heinemann, 2015. https://doi.org/https://doi.org/10.1016/B978-0-08-099995-1.00009-9.

[29] Aftosmis, M., Berger, M., and Murman, S., "Applications of space-filling-curves to cartesian methods for cfd," *42nd AIAA Aerospace Sciences Meeting and Exhibit*, 2004, p. 1232.