Launch Vehicle Ascent CFD for the Space Launch System

Derek J. Dalle*, Stuart E. Rogers†, Jamie G. Meeroff‡, Aaron C. Burkhead§,
NASA Ames Research Center, Moffett Field, CA 94035

Daniel G. Schauerhamer¶, Joshua F. Diaz‖,
Science and Technology Corp., Moffett Field, CA 94035

This paper will discuss the use of Computational Fluid Dynamics (CFD) for the Space Launch System (SLS) Program to model the ascent phase of flight. The ascent phase begins shortly after the vehicle clears the launch tower and extends to the first staging event. To model SLS's ascent, over one thousand numerical solutions of the Navier-Stokes equations are solved, and this analysis has been repeated for five different SLS configurations. To manage this demanding ascent CFD task, the SLS program has developed the Computational Aerosciences Productivity & Execution (CAPE) software, which is introduced in the paper. The paper also discusses some of the ways that CFD and high-end computing have advanced in the last decade and offer some comparisons to CFD used in the Space Shuttle Program.

I. Introduction

Launch vehicle aerodynamics is an area of Computational Fluid Dynamics (CFD) that has important engineering and historical implications but is arguably underrepresented in both publications and academia. This paper explains a subset of the launch vehicle CFD work that has been used to support NASA's Space Launch System (SLS) vehicle and uses this example to demonstrate some of the unique and important challenges that launch vehicles present to computational aerodynamicists. The first SLS vehicle flew on November 16, 2022 in support of NASA's Artemis I mission, the first of many missions aimed at returning humans to the surface of the moon and establishing a long-term presence. In particular, this paper focuses on the phase of flight called "ascent" within the SLS Program, which begins about 30 seconds after liftoff[a] and ends at the first separation event[b].

Though it may be surprising given the relative lack of research publications on launch vehicle CFD, launch vehicles have been one of the key applications driving the development of CFD [1,2]. Supporting the needs of the Space Shuttle Program in particular helped encourage the development of OVERFLOW [3, 4]. In many ways this work for SLS follows directly from the Space Shuttle ascent CFD progress as documented for example in [5]. Like any aerospace engineering system, each launch vehicle is unique and has its own requirements for a practical ascent CFD database. There are, however, some common characteristics that are true for SLS and likely to be applicable to most launch vehicles:

1. The databases are **high-volume**, meaning that a large number of individual CFD solutions are required.
2. Run matrices are **wide-ranging**, covering subsonic, transonic, and supersonic flight.
3. Each CFD simulation is used for multiple analysis products such as overall forces, large-scale structural loading, localized aerodynamic loads, and more.
4. Aerodynamic performance is not a major factor in the overall success of the vehicle.
5. **Uncertainty quantification** of aerodynamic outputs may be as or more important than the nominal values.

*Aerospace Eng., Computational Aerosciences Branch, Member AIAA, derek.j.dalle@nasa.gov
†Aerospace Eng., Computational Aerosciences Branch, Associate Fellow AIAA, stuart.e.rogers@nasa.gov
‡Aerospace Eng., Computational Aerosciences Branch, Member AIAA, jamie.g.meeroff@nasa.gov
§Aerospace Eng., Computational Aerosciences Branch, Member AIAA, aaron.c.burkhead@nasa.gov
¶Research Scientist/Eng., Computational Aerosciences Branch, Member AIAA, daniel.g.schauerhamer@nasa.gov
‖Research Scientist/Eng., Computational Aerosciences Branch, Member AIAA, joshua.f.diaz@nasa.gov

[a]The Mach 0.5 mark, which occurs at about 30 seconds for many launch vehicles, is a common selection. More generally "ascent" start when surface winds no longer cause large angles of attack on the vehicle, and it follows the phase called "transition."

[b]For some launch vehicles, for example with small strap-on boosters, "ascent" may extend beyond the first separation event.

6. The CFD solutions are for **static aerodynamics** only since compute capacity and algorithms cannot yet support a high-volume run matrix of scale-resolving CFD for buffet, aeroacoustics, etc.

The nature of a space launch vehicle, that it must accelerate from standing on the launch pad to orbital velocity as rapidly as possible, means that there is no cruise condition to focus the majority of the CFD analysis. Instead, the entire ascent phase may be almost uniformly important. In addition, strong wind shear at high altitude, combined with vehicles that are almost always aerodynamically unstable, mean that the total angle of attack (the angle between the direction the vehicle is pointed and the velocity vector) can be both significant and random. Therefore a database must cover all of the important Mach numbers, angles of attack, and sideslip angles that may be possible during flight, and this is the reason for characteristics 1 and 2 described earlier.

Characteristic 3 is fairly straightforward; at a high level the CFD solutions may be used for both integrated and distributed aerodynamic loads. SLS-specific details about this characteristic are presented in Sections II and V. A consequence of this characteristic is that the CFD cases are heavily "instrumented," meaning that the solvers are prepared in such a way that many different output products can be written to file.

The fourth characteristic stems from the very large thrust-to-drag ratio necessary for a space launch vehicle. Aerodynamic loading, and the structural response to it, may significantly affect launch availability because it determines at how high of total angle of attack the launch vehicle can fly during ascent. In particular, if the vehicle can withstand higher angles of attack, it can launch on windier days (to be specific, days with higher wind shear) and therefore is able to fly a larger percentage of the time.

Some of the critical challenges for ascent aerodynamics and launch vehicle aerodynamics in general come from the importance of uncertainty characterization and, where possible, uncertainty quantification, the fifth characteristic. Uncertainty quantification is particularly important for launch vehicles due to the lack of flight testing available in most cases. Because launch vehicles are often very high-value systems, and each piece of hardware is often single-use, understanding the accuracy of aerodynamics before the first flight test plays an even more important role than usual. An example showing some of the extra work taken to quantify uncertainty for SLS can be found in [6].

Finally, the sixth characteristic underlines that due to the high-volume run matrices discussed before, ascent aerodynamic CFD solutions typically only cover static aerodynamics. Important aspects of unsteady launch vehicle aerodynamics, including buffet and acoustics, in most cases exclusively rely on wind tunnel testing. While some unsteady CFD solutions can provide valuable information for these products, compute capacity is not yet sufficient to combine those techniques for a run matrix of 1000 or more cases [7]. The ability to run an entire run matrix of some variety of scale-resolving CFD and thus extract both steady and unsteady aerodynamics from the same data set is a long-term vision for ascent CFD.

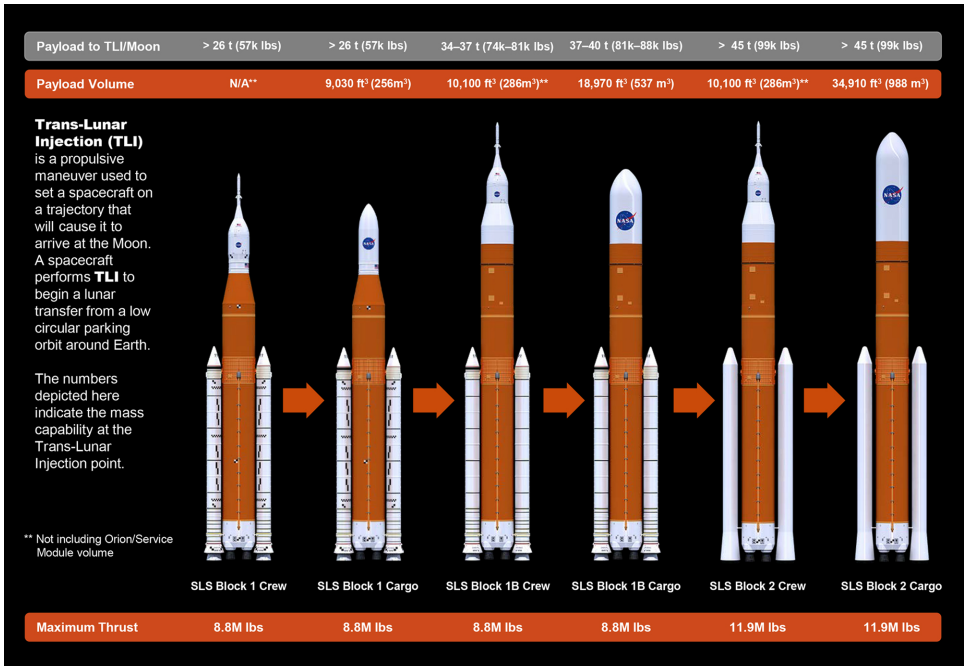


Figure 1. Six SLS configurations, or blocks, and summary of their capabilities

Another aspect of launch vehicle CFD is that a single program may develop several different vehicle versions suitable for different missions. One reason for this strategy is that many launch vehicles, including SLS, are primarily expendable, and so it is natural to incorporate improvements in each new vehicle and/or tailor a vehicle to the exact needs of a specific mission. Figure 1 shows the 6 main configurations that have been under consideration for SLS launches by NASA. Although only the three crew versions are on the current Artemis program manifest, the SLS program has done considerable work on all of them. Full ascent CFD analysis has been done on all four of the Block 1 and 1B vehicles, in some cases multiple times. Though each vehicle contains obvious similarities, they are all different enough to require separate aerodynamics databases. Therefore for SLS, and realistically many other space launch vehicles, establishing reliable and repeatable processes for ascent CFD is tremendously beneficial.

Section II talks about the current uses of ascent CFD, and Section III talks about additional applications that are feasible with the current state of the art. A view to the longer term advances is summarized in Section V, which wraps up with a brief summary of some of the historical impacts of launch vehicle CFD.

II. Current State of Ascent CFD for SLS

Since the most recent survey of SLS ascent aerodynamics [8], several significant advances have been made in ascent CFD procedures. The main changes since 2015 for the authors of this paper adding the use of a second CFD solver, NASA's unstructured solver FUN3D [9] alongside the continue use of NASA's structured overset solver OVERFLOW [10]. In addition, the Ames SLS CFD Team developed and released to the public the CAPE software package for controlling large CFD run matrices for complex configurations and handed over control of the ascent CFD run matrix from NASA-only internal tools to this new CAPE package. See Section IV for more information on CAPE and its use within the SLS program. All CFD solutions studied here were performed on the *Pleiades*, *Electra*, and *Aitken* supercomputers at the NASA Advanced Supercomputing (NAS) facility.

A. CFD Procedures: FUN3D

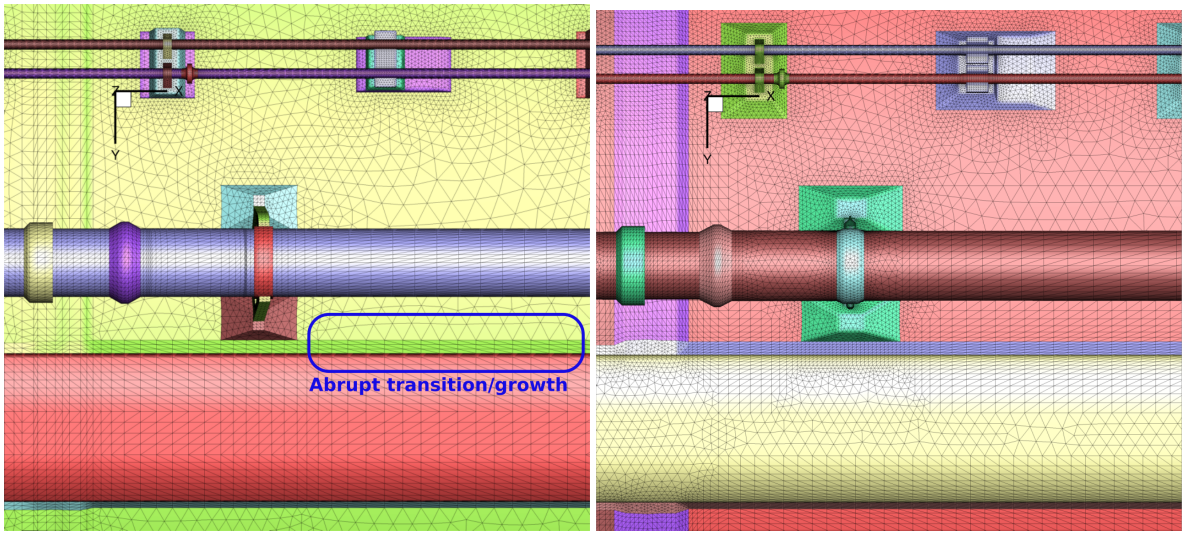
FUN3D is a fully unstructured solver for the 3-D Navier-Stokes equations developed at NASA Langley Research Center. For SLS ascent CFD, versions 13.1 [11] through 13.7 [9] have been used, with a modification to newer versions in order to write surface output in `.plt` format instead of `.szplt`[c]. These `.plt` files are used to create databases of surface solutions as described in subsections D, E, and F. The solutions are Reynolds-averaged Navier-Stokes (RANS) solutions with turbulence closure from the one-equation Spalart-Allmaras [12] turbulence model without rotational corrections or quadratic constitutive relation (QCR). The flow is considered turbulent everywhere. Non-slip, cold walls with the default FUN3D temperature were used to model the surface boundaries of the vehicle except for the flow-through faces in the plenums of the six nozzles where simulated fuel and oxidizer enters the flow. These faces use total pressure and subsonic pressure[d] to set the thrust and other conditions for each individual engine. The throat and subsonic regions of each nozzle have been modified to match thrust and mass flow but not temperature. To reduce cost, ascent CFD simulations use the perfect-gas version of FUN3D. This means that the exhaust is modeled as high-temperature air and cannot match all three of thrust, mass flow rate, and temperature.

Two or three (three for the most recent CFD configurations) feature-based mesh adaptation cycles are performed for each ascent CFD case. Unstructured meshes with triangular-prism layers cells and mostly tetrahedral volume cells are generated using AFLR3[e] with a wall spacing ensuring $y^+ \leq 1$ for the highest-Reynolds number flight condition in the trajectory.

For FUN3D, a case is started with a nominal sequence of 3000 steady-state RANS iterations, although recently a procedure with 2500 steady-state iterations has been implemented. The CAPE interface to FUN3D, simply called `pyfun`, makes stopping and restarting FUN3D multiple times convenient, and so the 3000-iteration approach runs FUN3D four separate times. In CAPE terminology, this means the setup has four "phases." Mesh adaptation is performed after the second and third phases in this scheme. For the newer, 2500-iteration approach, five phases are used, with mesh adaptations after the second, third, and fourth phases.

If after these nominal iterations the iterative histories are determined by a human analyst to be unsteady, the case is restarted with unsteady RANS iterations for a nondimensional type step of 1.0 and 5 subiterations. If the case does not have oscillatory features but is still changing, additional steady-state RANS iterations are prescribed. This judgment is continued until a trustworthy average (for unsteady cases) or fixed asymptotic value (for steady cases) is observed in the iterative histories of forces & moments.

[c] Getting the `.plt` output format requires the `--plt_tecplot_output` option when calling FUN3D's solver.
[d] This is FUN3D boundary condition 7011.
[e] https://www.simcenter.msstate.edu/software/documentation/aflr3/index.html



a) SLS-10008 (Block 1) mesh from 2017 **b)** SLS-28008 (Block 1B) mesh from 2022

Figure 2. Two versions of unstructured surface mesh on lower side of LH2 tank and Core Stage intertank

The unstructured surface meshes for SLS ascent CFD have been created using a software package called ANSA[f]. The SLS team continues to use this software due to its configuration control capabilities and ability to do CAD fixes and surface mesh generation using a single tool. Two versions of unstructured surface meshes on the same region of the SLS core stage are highlighted in Fig. 2. Significant improvements have been made since the initial set of FUN3D results from 2017. The original mesh, shown in Fig. 2a has instances where high–aspect-ratio triangles are adjacent to isometric triangles, which results in a large jump in length scale in one direction. A blue box and text highlights one of these regions in Fig. 2a, where the triangles on the main acreage of the liquid hydrogen tank (yellow surface) has triangles much larger than the immediately adjacent systems tunnel (green and red surfaces). In Fig. 2b, these transitions are much smoother.

The capability to produce the higher-quality mesh approach demonstrated in Fig. 2b has always existed, but the amount of work required to produce this kind of unstructured mesh was judged to be unavailable in the 2017 versions of these meshes. Figure 2 also shows some differences in the geometry; in general the SLS Core Stage has added more thermal protection system (TPS) to its outer mold line as the design matured.

For each volume grid, a surface mesh exported from ANSA is the primary input to AFLR3, which creates the initial volume mesh. Figure 3 shows an example of a final mesh after three adaptation cycles performed by FUN3D on the Block 1 configuration at Mach 1.1 at 4° angle of attack and 0° sideslip. The cut plane shown in Fig. 3, demonstrates how the adaptive mesh refinement conforms to the shocks in this case. Coloring of the vehicle surface in Fig. 3 is by pressure coefficient, with blue for negative c_p, and the cut plane is colored by the local Mach number, with green and orange less than and greater than the freestream Mach number, respectively. After adaptive mesh refinement, the range of nodes seen for various SLS configurations is 50 to 95 million.

A region of finer mesh about one diameter away from the vehicle's surface is visible in Fig. 3. This is a remnant of an extra surface that is added to the AFLR3 input surface that allow users to have more control over the mesh resolution of the initial volume mesh. Figure 4 shows an example of the iterative history of the global density L_2 error residual for a Block 1 simulation at Mach 1.75 for the updated three-adaptation approach. Four spikes in the residual occur, which are annotated in Fig. 4.

The surface triangulations for FUN3D have ranged in size from 1.0 to 1.6 million nodes and 2.1 to 3.3 million triangles. The unstructured surface meshes also contain a small number of quads on the six engine boundary condition faces. Initial volume meshes, prior to mesh adaptation in FUN3D, have had 53 to 78 million nodes.

B. CFD Procedures: OVERFLOW

OVERFLOW is a structured-overset grid CFD solver developed at Langley Research Center and has a long history of support for NASA's missions [10]. The ascent CFD OVERFLOW solutions have utilized versions 2.2G through

[f] https://www.beta-cae.com/ansa.htm

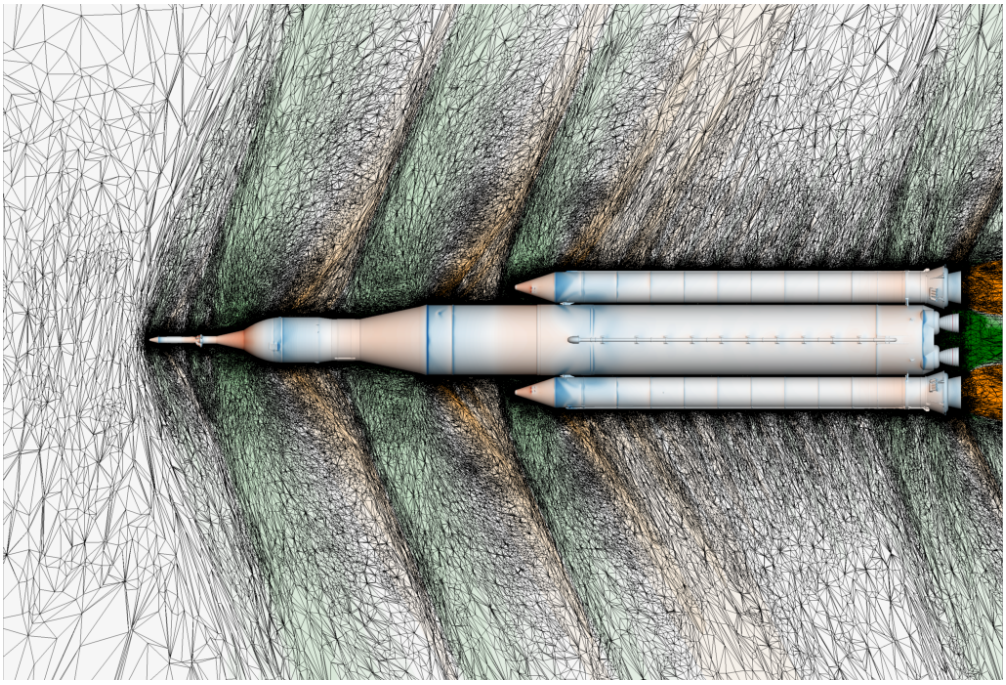


Figure 3. SLS Block 1 flow solution from Mach 1.1, 4° angle of attack on z=0 cut plane

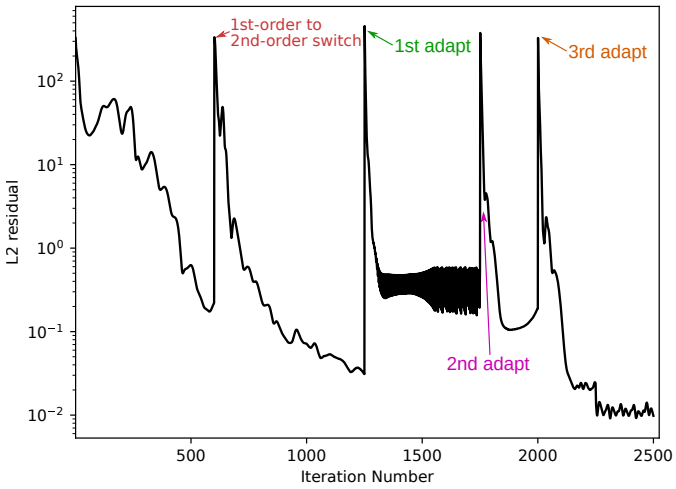


Figure 4. Representative iterative history of L_2 residual for a FUN3D ascent CFD case at Mach 1.75, with annotations

2.2L and 2.3, and the team is implementing 2.4 for the remaining ascent CFD run matrices. Solutions from after 2014 include the Cartesian off-body grid adaptation capability [13].

For OVERFLOW, a case is started with a nominal sequence of 20,000 steady-state RANS iterations. In CAPE terminology, these 20,000 iterations are run in four "phases," with transitions to the next phase at 6000, 8000, and 10,000 iterations. The first phase is run with "full multigrid" cycles using the `FMG` option in the input namelist. In particular, 2000 iterations on each of three multigrid levels are used. Using starting iterations on coarser grids helps accelerate the initial transients when starting from freestream flow. In the second phase, from 6000 to 8000 iterations, an off-body mesh adaptation is performed every 100 iterations, and in the third phase this is reduced to an adaptation every 250 iterations. Then the fourth and final steady-state phase is a long phase of 10,000 iterations with an off-body mesh adaptation performed every 500 iterations. The structured overset grid systems used for SLS ascent CFD have typically had just over 1800 near-body grids, with about 4 million points in the integrated surface mesh. The initial near-body volume meshes, not counting the off-body grids, have between 540 and 610 million points. With off-body

grids and OVERFLOW mesh adaptations, the grid sizes grow to a range of 600 to 671 million points.

If necessary, in particular if the force & moment iterative histories show unsteady behavior other than initial startup transient, an additional two phases are added with unsteady RANS inputs. The first phase of 2000 iterations have an off-body mesh adaptation every 250 iterations, which allows the mesh to adjust to changes between steady-state and unsteady flow solutions. The second phase also has 2000 iterations but no mesh adaptations. Both unsteady phases use a nondimensional physical time step of 1.0 and 5 subiterations. The second phase is repeated until the iterative histories reach a limit cycle so that the average of the last 2000 iterations is stable.

To extract data for surface pressures (subsection D), line loads (subsection E), and protuberance air loads (subsection F), the surface grid system is projected to a unique triangulation using either `mixsur` or `usurp` from the Chimera Grid Tools package [14]. The sizes of these triangulations are important since they determine the size of the surface pressure database. For existing SLS ascent CFD grid systems, these triangulations have had 6.6 to 7.3 million nodes and 13.2 to 14.5 million triangles.

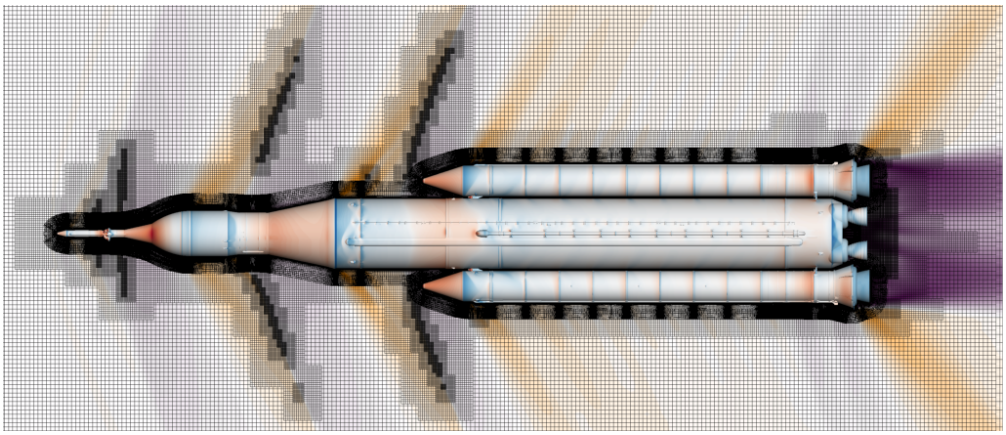


Figure 5. SLS Block 1 OVERFLOW flow solution from Mach 1.1, α_p=4°, ϕ_p=30° on z=0 cut plane

Figure 5 shows a view of an OVERFLOW solution very similar to the FUN3D solution in Fig. 3. Compared to Fig. 3, this is at a slightly different flow condition (with 2° sideslip instead of 0°), purple replaces the role of green, and the engine plumes are not modeled. The off-body mesh refinement has a very different character to those of the unstructured mesh refinement, but the results are consistently similar.

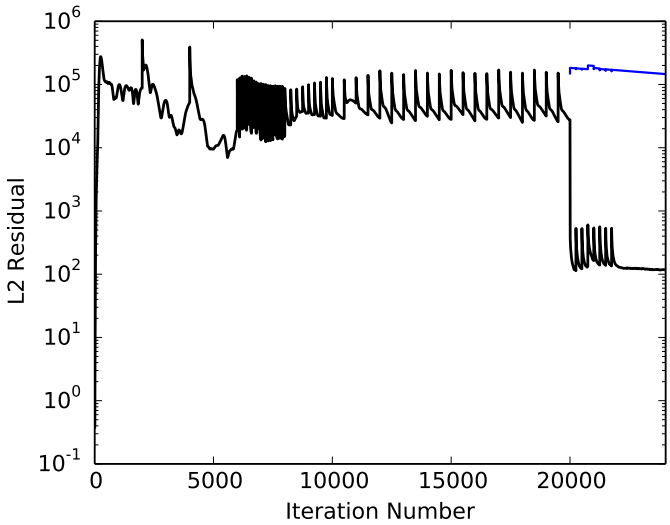


Figure 6. OVERFLOW global density L_2 residual history for SLS Block 1, Mach 1.1, α_p=4°, ϕ_p=30°

Figure 6 shows the iterative residual history for the same case as shown in Fig. 5. The early spikes in residual are due to phase transitions, and the more frequent smaller spikes throughout the middle portion show each time the mesh is adapted. The blue line for the final 6000 iterations shows the residual prior to the subiterations at each time-accurate

time step; the overall residual only drops significantly after the dual time-stepping time-accurate approach is used.

C. Run Matrix

A key aspect of a successful and valuable ascent CFD data set is the design of a good run matrix. Figure 7 shows a slice through the run matrix at a fixed Mach number for SLS ascent CFD databases. Each dot represents one of the 69 cases run by the SLS teams at each Mach number, and annotations in Fig. 7 show the outer edges of different databases that use ascent CFD data. These databases are described in the following subsections. The gray coloring in the background of Fig. 7 are contours of the integrated vehicle rolling moment coefficient (*CLL*) for one Mach number, which is selected as an example that demonstrates the quality of interpolation from this run matrix.

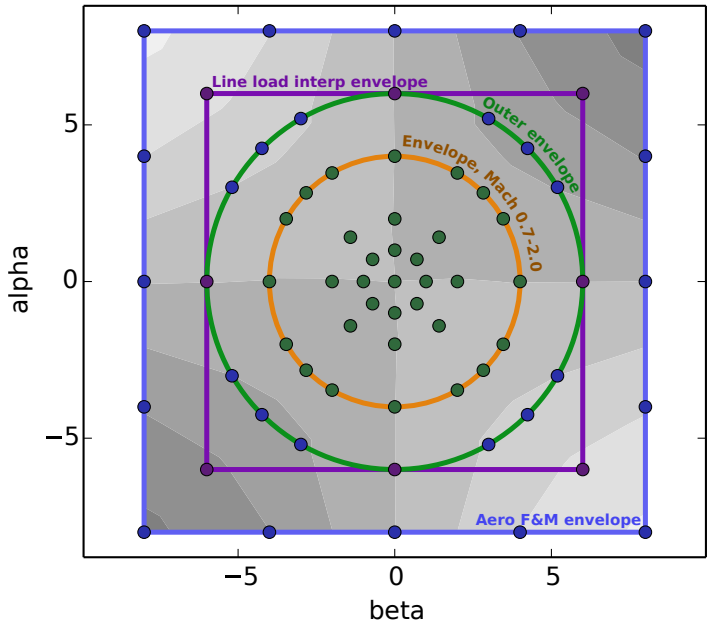


Figure 7. Annotated SLS ascent CFD run matrix for fixed Mach number

The example run matrix slice shown in Fig. 7 has two separate apparent shape features to it: squares and circles. Total angle of attack, or α_p, is the angle between the vehicle's x body axis (the direction the vehicle is pointing) and the velocity vector. This angle is a natural choice for launch vehicles since they tend to have approximate radial symmetry. A complementary angle ϕ_p for a given dot in Fig. 7 is just the angle between the positive-α axis and the line connecting the the origin and that dot. This means that each point in Fig. 7 can be described as either a pair of angle of attack and sidelip angle, (α,β), or total angle of attack and roll, (α_p,ϕ_p). For example, consider two vehicles, with one at +4° angle of attack and the other at −4°, and both with zero sideslip. Both vehicles have $\alpha_p = +4°$ total angle of attack, but the first has a roll angle of ϕ_p=0° while the second has ϕ_p = 180°. A third vehicle with α=0° angle of attack and β=4° sideslip would still have 4° total angle of attack but ϕ_p=90°.

Picking run matrix points based on α_p and ϕ_p is natural for launch vehicles since their rough radial symmetry usually leads to a bounded maximum value of α_p and a mostly random ϕ_p. However, interpolation in $\alpha_p\phi_p$-space is challenging, so it is convenient to add some extra cases with nice values of α and β so that the run matrix has an overall square shape. Then the entire interior can be interpolated using α and β. To describe these two pairs of angles, it is easiest to start with the components of the vehicle's relative velocity in its own body-coordinate system. Let U be the magnitude of the vehicle velocity relative to the wind, and the three components be u, v, and w such that a positive u is the nose-forward velocity component, positive v is the velocity to the pilot's (or imaginary pilot's) right, and positive w is the downward velocity (meaning the windward side is on the bottom of the vehicle). The three components are

easily described in terms of the two angle pairs as follows:

$$u = U\cos\alpha\cos\beta \qquad u = U\cos\alpha_p \tag{1}$$
$$v = U\sin\beta \qquad v = U\sin\alpha_p\sin\phi_p \tag{2}$$
$$w = U\sin\alpha\cos\beta \qquad w = U\sin\alpha_p\cos\phi_p \tag{3}$$

The relationship between the two angles, then, is somewhat indirect. By convention, $\alpha_p \geq 0^\circ$ is always nonnegative, and $0^\circ \leq \phi_p < 360^\circ$ The conversion from (α,β) to (α_p,ϕ_p) is shown in Eq. (4), and the reverse conversion is shown in Eq. (5). Both directions utilize the two-argument arc-tangent function common in many programming languages.

$$\alpha_p = \cos^{-1}(\cos\alpha\cos\beta) \qquad \phi_p = 180^\circ - \tan^{-1}(\sin\beta, -\cos\beta\sin\alpha) \tag{4}$$
$$\alpha = \tan^{-1}(\sin\alpha_p\cos\phi, \cos\alpha_p)\&\beta \qquad = \sin^{-1}(\sin\alpha_p\sin\phi_p) \tag{5}$$

Note from these formulas that (α_p,ϕ_p) is **not** the same as polar coordinates of (α,β), although the errors of such an assumption might be insignificant for small angles.

D. Surface Pressure Database

The surface pressure database is both the largest (in terms of file size) and simplest to understand of the three primary ascent CFD products. In the simplest form, it is a collection of surface pressure fields from some or all of the CFD cases in the run matrix. For SLS not all CFD solutions are included in all of the databases, and the surface pressure database only includes the cases inside the green circle in Fig. 7, resulting in about 931 total cases. For FUN3D, this is a subset of the `.plt` file written by FUN3D of the surface solution, and for OVERFLOW this is a conversion of the `grid.i.triq` file produced by `usurp` or `mixsur` and then converted to Tecplot® `.plt` format. Later versions of the surface pressure database have been developed as Python[g] packages. The packages have a smaller total file size since each case contains the same triangle node coordinates and node indices, and repeating them for every individual CFD solution is unnecessary. Creating a programmable interface to the surface pressure enables other advanced features such as interpolation and dispersion, but discussion of those capabilities is left to Section V.

At present, SLS surface pressure databases include the pressure coefficient,

$$c_p = \frac{p - p_\infty}{q_\infty} \tag{6}$$
$$q_\infty = \frac{1}{2}\rho_\infty U_\infty^2 = \frac{\gamma}{2}p_\infty M_\infty^2 \tag{7}$$

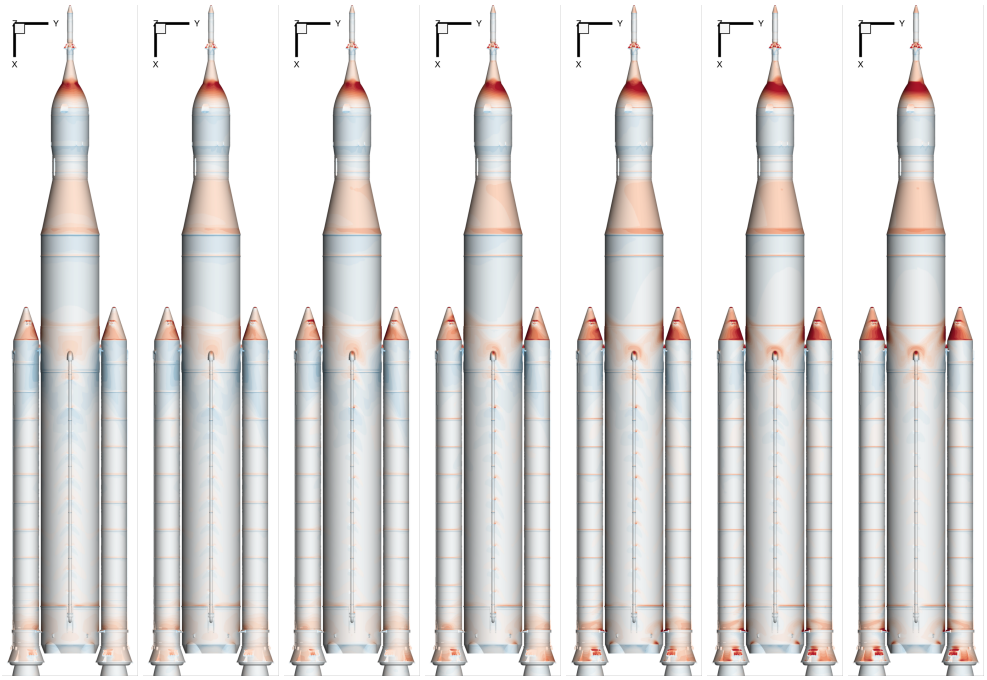
only, and not other state variables like skin friction coefficient components or static temperature. Adding these would not dramatically alter the nature of a surface database. Figures 8 and 9 show images of a subset of one of these surface pressure databases, specifically a SLS Block 1 surface pressure database from FUN3D. Both figures show the same 7 Mach 1.75 cases, specifically 7 of the 9 cases on the orange circle in Fig. 7 with $\beta \geq 0^\circ$. Figure 8 shows a top view of these seven cases, while Fig. 9 shows a view from below the vehicle. Surfaces with blue color show a negative pressure coefficient, and red surfaces have positive pressure coefficient.

E. Line Loads

In the SLS program, there are two major types of distributed loads: surface pressures and line loads (sometimes also called sectional loads). The surface pressure database contains the pressure coefficient at all points on the vehicle for all expected flight conditions and contains a large amount of data. The line load database is essentially a projection of this database into one or several 1-D loads. This means dividing the vehicle into a number of slices, as shown in Fig. 10, and calculating the load on each slice. The line loads do include viscous forces, so technically the line load database differs slightly from a projection of the surface pressure database. These line loads are calculated using the `triloadCmd` tool [15], which is part of Chimera Grid Tools [14].

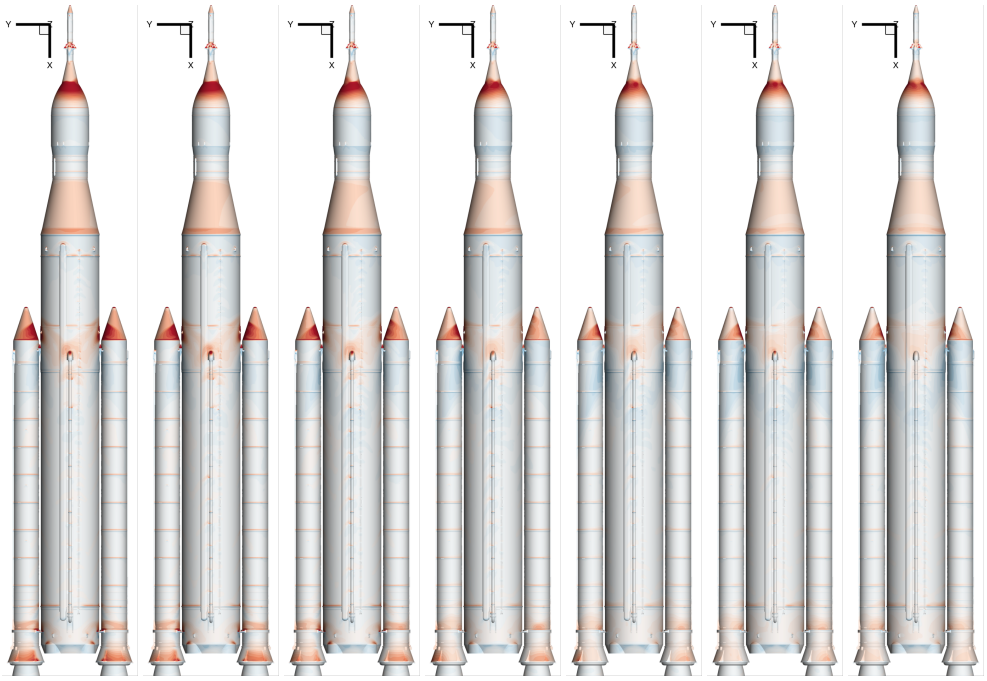
Another particular for the SLS program line load databases is that there are four separate line loads calculated for each CFD solution. First the entire integrated stack is divided into 200 slices, as shown in Fig. 10a, and then the vehicle is split into three separate bodies of 200 slices as shown in Fig. 10b. For SLS, the three separate line loads are the main products used by the loads team to estimate structural loads; this is logical since the three bodies structurally are relatively independent. The data extracted from a single CFD solution is then the 6 force & moment coefficients

[g] https://www.python.org



a) ϕ_p=0° **b)** 30° **c)** 60° **d)** 90° **e)** 120° **f)** 150° **g)** 180°

Figure 8. Subset of SLS Block 1 surface pressure database at Mach 1.75, α_p=4°, top view



a) ϕ_p=0° **b)** 30° **c)** 60° **d)** 90° **e)** 120° **f)** 150° **g)** 180°

Figure 9. Subset of SLS Block 1 surface pressure database at Mach 1.75, α_p=4°, bottom view

a) Common slices of Integrated Vehicle

b) Separate slices of three main bodies

Figure 10. Sectional load slices for SLS Block 1B configuration

for each of the 800 slices shown in Fig. 10. Contributions from the moment coefficients are typically negligible[h]. Technically, the values in the line load databases are instead the derivatives of the force (& moment) coefficients with respect to nondimensional axial coordinate (i.e. x/L_{ref}). Using this derivative avoids the problem of the database values being directly affected by the slice size. The bounds for the SLS line load databases are $-6^\circ \leq \alpha \leq 6^\circ$ and $-6^\circ \leq \beta \leq 6^\circ$, which corresponds to the purple square in Fig. 7. However, this database is not delivered at the 53 points inside the purple square of Fig. 7 directly; instead the database contains a separate line load at every combination of integer α and β, which is 169 total points for each Mach number. Therefore the line load database is first interpolated onto this regular $\alpha - \beta$ grid before delivery to loads teams or other users. As a result some of the blue points in Fig. 7 that are outside the purple circle actually affect the line load database despite being outside the $\alpha - \beta$ database limits.

Users may wonder what the point of such a database is because the surface pressure database includes all of the same data and more (at least if the skin friction coefficients were added to the surface database). The explanation is that using 1-D distributed aerodynamics instead of 2-D greatly simplifies the use of the database. First, the number of scalar loads from a single CFD solution for the line load database is about 2400, as described above (4800 including the moment line loads), whereas the surface pressure database requires at least 1 million. Furthermore, each surface pressure database has a different size depending on the CFD mesh used to compute solutions, which complicates the use of multiple versions of the database, whereas the line load database can be specified to contain a precise number of slices that remains constant for all databases. Finally, interpolation in Mach number, α, and β is much more challenging (though entirely possible) for a surface pressure database. Therefore line loads are a useful simplification, but only if the design of the vehicle makes this one-dimensional loading a reasonable approximation. For launch vehicles, the applicability of this type of model is common, and line loads are a widely adopted technique for communication between aerodynamics and structural analysis.

Figure 11 shows an example of the normal-force line loads on the right Solid Rocket Booster (RSRB) from the same 7 cases shown in Fig. 8 and 9 plus two more cases. These conditions are all at Mach 1.75, 4° total angle of attack, and nonnegative sideslip. The two additional cases (compared to Fig. 8) are at ϕ_p=45° and 135°. Plotting line loads from multiple solutions simultaneously like this allows analysts to check for out-of-family results and is a proven method for finding CFD cases that need further improvement. In this way, line loads have also become a critical aspect of quality assurance for SLS, as the aerodynamics team looks at many versions of these 9-case plots at each Mach number (often in four groups, nonnegative and nonpositive subsets of both α and β) and for each force coefficient in order to find cases that do not have the expected symmetries.

F. Protuberance Air Loads

Protuberance air loads, which in some cases has also been called excrescence loads, is a database of loads on various parts of the launch vehicle other than the main structure of the core and boosters. Protuberances are present on all parts of the launch vehicle, including Orion, the upper stage, the core stage, and both boosters. Examples of parts covered

[h]The moment coefficients are mostly the force coefficients times each slice's distance from the moment reference point. For infinitely thin slices, the moment line loads would be completely trivial.

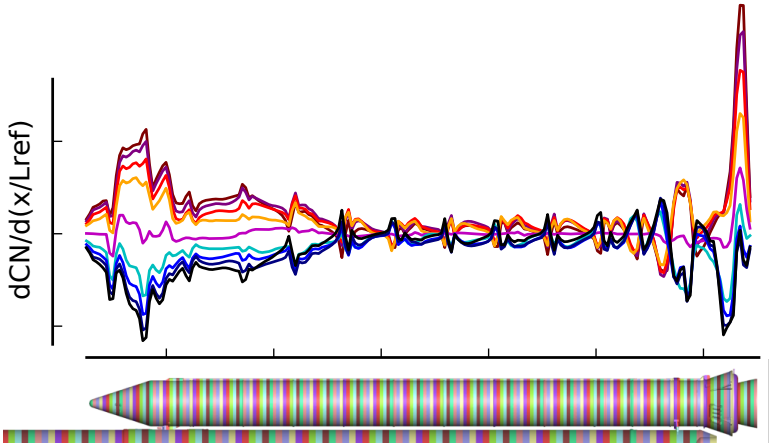


Figure 11. Family of 9 line loads for Block 1 RSRB at Mach 1.75, 4° total angle of attack colored by angle of attack, blue for $\alpha < 0$

in the SLS protuberance air loads database include:

- 7 cameras,
- 7 or 8 (depending on the version of SLS) systems tunnels,
- 2 liquid oxygen feed lines,
- 2 fuel tank pressurization lines,
- hundreds of support brackets for other parts like the feed lines,
- and many others.

The loads on some of these parts (for example the 7 cameras) are delivered as integrated forces & moments on a series of "patches" such as front face, left face, etc. Other parts, including the systems tunnels and feed lines, are large enough that distributed loads are needed, so the loads on these parts are delivered as line loads. From a simplistic standpoint, this database can be characterized as a miscellaneous loads database.

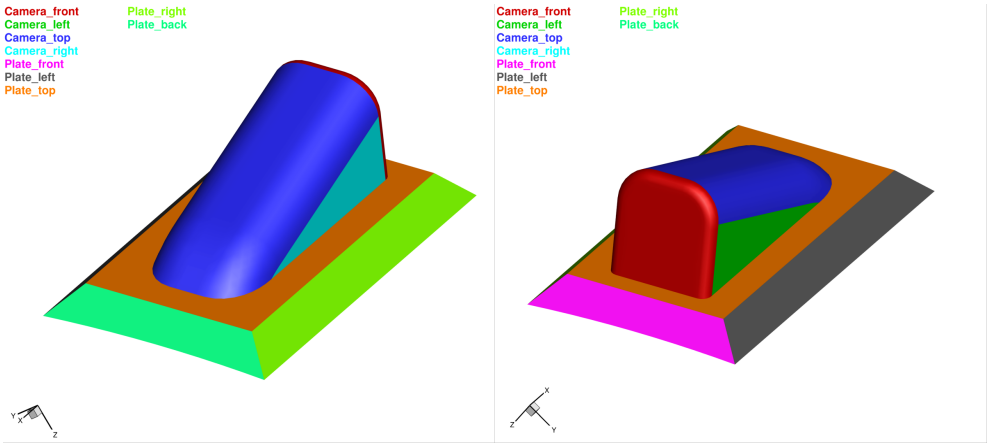
For the protuberances where integrated forces & moments (rather than distributed loads) are appropriate, the loads for the SLS program are further broken into patches. Examples of these patches on two SLS protuberances are shown in Fig. 12. There are two main reasons for delivering loads on these separate patches:

- it provides the ability to calculate loads on sub-assemblies without additional consultation with the CFD team, and
- it enables generation of extra conservatism by, for example, taking the largest left-ward force on one patch and combining it with the largest right-ward face on another patch.

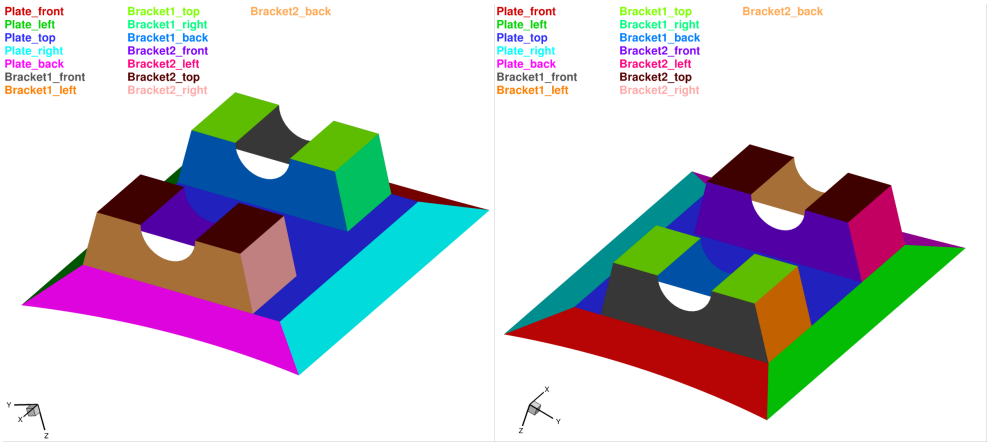
The benefit of the second of these points is somewhat debatable since it effectively certifies a structure against a load that the aerodynamics never actually predicts, but it has been an occasional practice in the SLS program.

Despite the varied nature of these loads, they are critical to ensure the safety and success of a launch vehicle. These structural features that at least partially protrude from the vehicle and are exposed to external aerodynamics are all designed to perform a function other than benefiting aerodynamics, and many of those functions are critical to a successful mission. Since features such as oxidizer feed lines, flight instrumentation assemblies, thrust vectoring braces, etc. are designed with aerodynamic loads at best a secondary concern, verifying that these parts can withstand the aerodynamic environment of flight is crucial.

Each protuberance load is analyzed in the local-body coordinate system. In this system, the normal force (*CN*) points away from the centerline (*x*-axis) or the centerline of the SRB it is attached to, the axial force (*CA*) points downstream toward the engines but tangent to the local surface, and the lateral force (*CY*) completes a right-handed system. As a result, a positive *CN* on the bottom-side liquid oxygen (LOX) feed line points in exactly the opposite direction as a positive *CN* on the integrated vehicle. A positive *CN* on the RSRB main tunnel, which is on the far right-hand side of the integrated vehicle, is parallel to the integrated *CY*. Among other benefits, using this local body



a) Core Stage engine section camera 2 of 4



b) CS LH2 tank press line bellows bracket 4 of 4

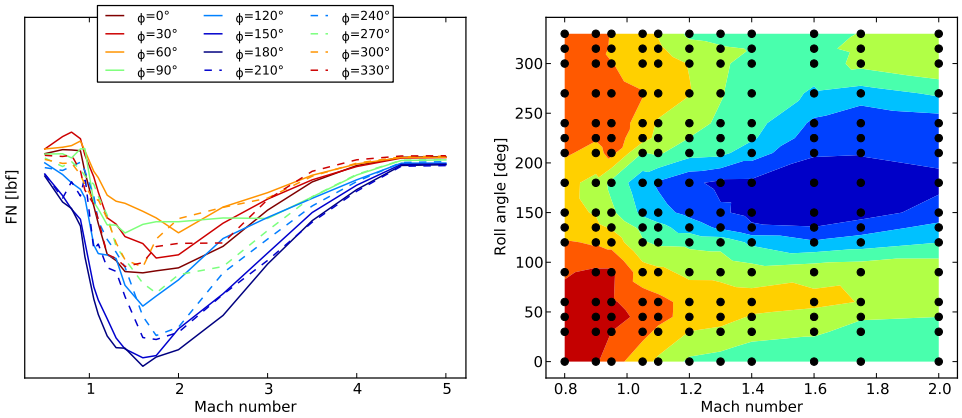
Figure 12. Examples of patches on two SLS protuberances

coordinates in this fashion makes it easier to compare different parts that are at different clocking angles (0° for a protuberance on the top of the vehicle, 90° for a protuberance on the left side, up to 359° for a part just right of the top center).

Because of the large number of protuberances and either a few or many patches on each protuberance, the integrated load portion of the database contains a huge number of loads. With that observation, quality assurance is both important and challenging. For databases in the SLS Program, a two-step process of quality assurance on these loads has been developed. The first step, demonstrated in Fig. 13, is to look at all the results for each individual protuberance. The two plots in Fig. 13 actually show the same data (except that 13a extends to Mach 5 whereas 13b stops at Mach 2) in order to give human eyes two different chances to catch problematic or unusual results. The second step is to look at envelopes of the minimum and maximum dimensional load as a function of Mach number for each family of protuberance. An example of this is shown in Fig. 14 for any of 23 Core Stage LH2 tank press line sliding bracket clamp to gaseous oxygen press line. This type of analysis condenses a large set of data into a small enough set that it is possible to compare multiple databases, and Fig. 14 indeed shows the results from three separate databases.

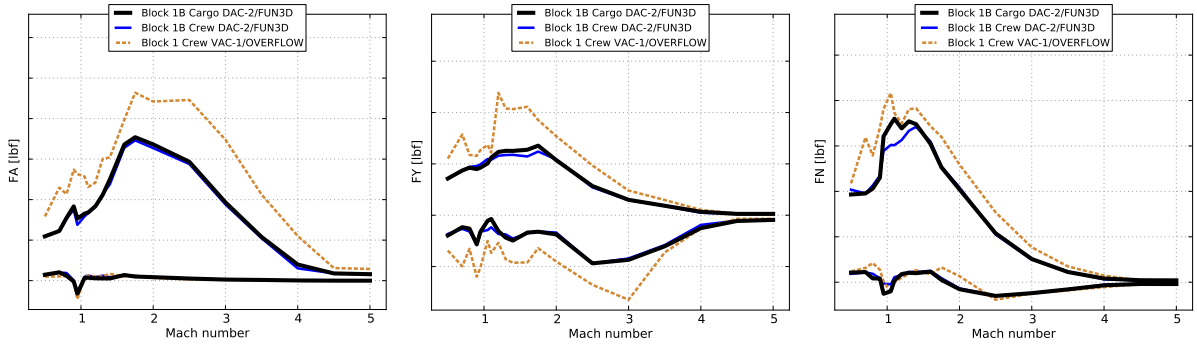
For both Figs. 13 and 14, the results have been dimensionalized using the nominal dynamic pressure from an appropriate trajectory for the matching SLS configuration. The difference in mass between Block 1B and Block 1 (without a corresponding significant difference in thrust) results in a lower dynamic pressure for Block 1B. Since Fig. 14 is dimensional, the lower dynamic pressure leads to lower load magnitudes, even if the coefficients match. Using dynamic pressure in this manner seems problematic at first since force coefficients are very well established in aerodynamics. However, launch vehicles during ascent encounter large differences in dynamic pressure, and this dimensional approach to Figs. 13 and 14 allows analysts to focus on loads where they actually challenge the structure.

Figure 15 shows two examples of the slices used to compute line loads on some of the SLS protuberances for



a) *CN* Mach sweeps at α_p=4° and various ϕ_p **b)** Contour plot of *CN* for all α_p=4° results

Figure 13. Normal force on RSRB aft booster separation motor pod

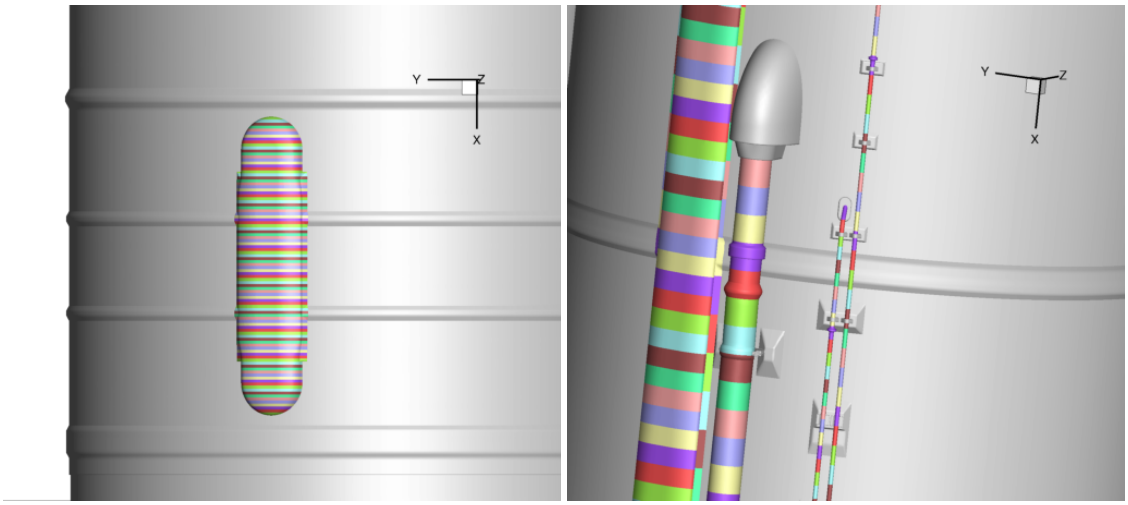


a) Axial force **b)** Side force **c)** Side force

Figure 14. Min and max loads for one SLS protuberance as a function of Mach number

which distributed aerodynamics is appropriate and necessary. For parts like these, the protuberance air loads consists of a database of the three components of force in the local protuberance body frame for each of the small colored slices pictured. So for example if a protuberance is divided into 100 slices, and the database contains about 1200 CFD solutions, the load database for that protuberance will have 120,000 values for each of the three force coefficient components. In most practical cases, this provides sufficient spatial resolution of the aerodynamic load to analyze the structural integrity of a part, especially since these parts are often thin compared to their length. However, the capability to deliver a full surface pressure database for a single protuberance surface (including the three components of skin friction coefficient) does exist within the existing SLS ascent CFD framework.

Figure 16 is an attempt to quantitatively summarize the load database for one protuberance requiring distributed aerodynamics. This figure takes one protuberance, the LOX feed line on the top side of the SLS Core Stage, and plots both the minimum and maximum load across an entire run matrix for each of the 100 slices of that feed line. Instead of plotting the minimum and maximum coefficient for each segment, what's shown is the dimensional load on each segment, which accounts for the rise and fall in dynamic pressure during ascent. Furthermore, 16 shows this envelope for three separate run matrices, corresponding to database on three configurations of SLS. The Block 1B loads tend to be smaller in magnitude because it is a larger and heavier vehicle than Block 1 but with the same or similar thrust until the larger upper stage takes over later in flight. SLS vehicles have over 150 protuberances in each version of this database, and the aerodynamics team inspects plots like either Figs. 13 and 14 or Fig. 16 for each one, depending on the protuberance type. As a result, the internal memos documenting each of these databases stretch to over 500 pages each. Finally, the users of these loads have additional heterogeneous considerations that vary for each protuberance. For example, the liquid oxygen feed lines also have to withstand the force of changing the direction of the high–flow-rate oxidizer within.



a) Exploration Upper Stage systems tunnel **b)** Several parts on Core Stage intertank and LH2 tank

Figure 15. Examples of slices for Block 1B protuberance line loads

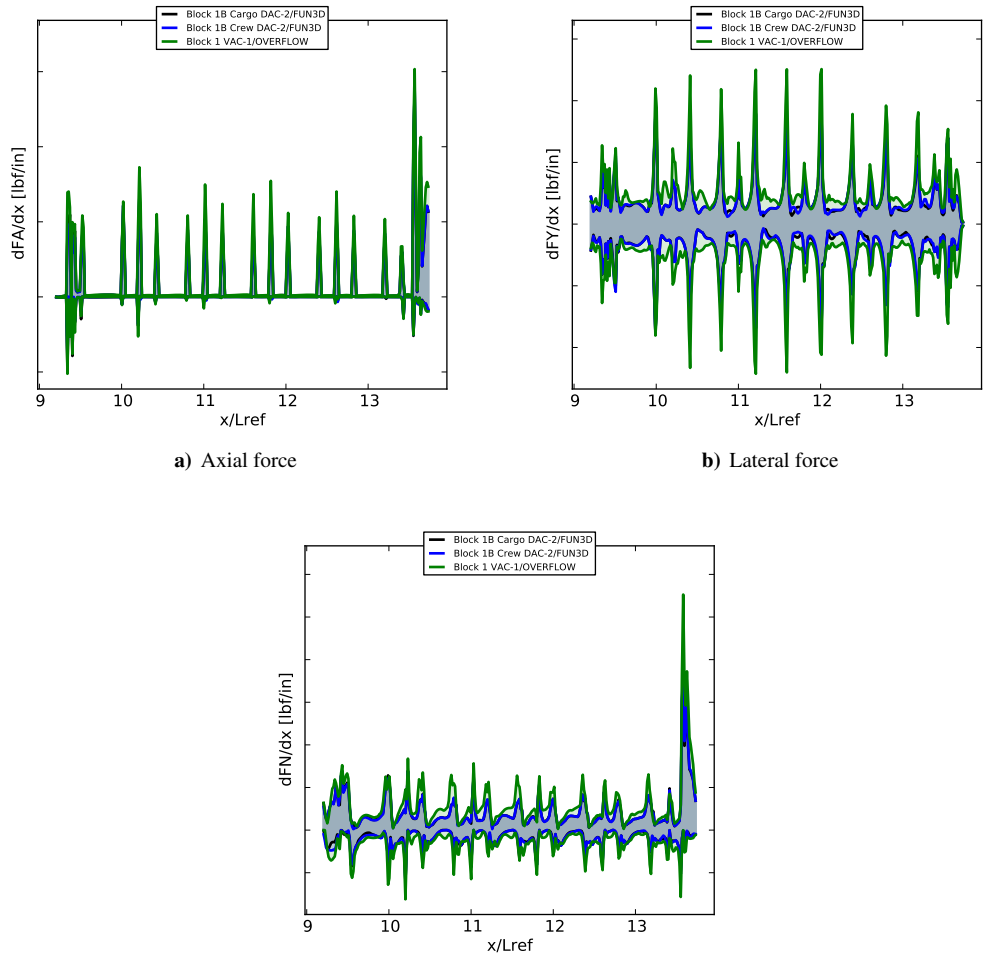
To compute these protuberance air loads, the SLS program analysts simply use commands from CAPE (see Section IV). The procedure is similar to the line load procedure (see Section II.E), taking a triangulated surface grid—either a Tecplot® file written by FUN3D or an output from either `usurp` or `mixsur` in the case of OVERFLOW. The user then provides a template file with the same geometry as the protuberance but with the individual patches labeled (this means the protuberance patches from Fig. 12 need not be broken out as separate components in the actual CFD grid). These patches must have the same geometry as the the actual surface grid (or close) but not necessarily the same discretization. Then CAPE uses a process referred to internally as "TriqFM" to map the template's patch identifiers onto the surface grid and then integrate the surface pressures and skin friction on each patch. CAPE also (optionally) writes a `.plt` format of the solution on just the protuberance in question with each patch in a separate zone, which opens the door to users using more detailed distributed aerodynamic loads without needing to interact with the full surface pressure database.

G. Force & Moment Uncertainty Quantification

The last application of ascent CFD covered in this paper is contributing to the uncertainty quantification of integrated force & moment. CFD is not used for the nominal forces & moments that develop the trajectory predictions and flight control development; those nominal aerodynamic products come directly from wind tunnel testing. The advantages of wind tunnel testing is that naturally there are no discretization errors, but at the same time CFD can model the physics of flight more directly than a wind tunnel (for example analyzing the rocket plumes, running at flight Reynolds number, and the lack of any mounting mechanism or wind tunnel walls). To bridge this divide and try to get the best of both worlds, CFD has not been used for the program's nominal forces & moments, but the difference between wind tunnel CFD and flight CFD is one of the terms incorporated into force & moment uncertainty quantification. In other words, CFD is used to estimate the modeling errors caused by the various physical limitations of the wind tunnel testing.

III. Additional Possible Applications of Ascent CFD

This section discusses ideas for expansions or expanded applications of ascent CFD for which most of the development has already been done, but where more testing and experience is necessary. For example,using CFD for the nominal values in the ascent force & moment database would be easy to do, but doing so would naturally put a lot of pressure on CFD to be correct. In other cases, extending the use of ascent CFD may require changes in procedure for other engineering groups (for example loads teams or guidance, navigation & control). However, none of these changes requires an increase in high-performance computing capacity.



a) Axial force

b) Lateral force

c) Normal force

Figure 16. Envelope of Core Stage top LOX feed line line loads for three SLS configurations

A. Forces & Moments

Using the integrated forces & moments from CFD for the nominal values in the database, which is used to estimate performance, develop control laws, etc., would require no extra development since this data is already extracted. Doing so would allow the database to be based on direct simulation of the flight vehicle, rather than developing the database with tests from a lower Reynolds number, and incorporate direct simulations of the launch vehicle engines. The challenge for this concept is two-fold: are the results accurate, and can the analysts build trust in the results? At this point, CFD results are getting more accurate, but there is not a clear path to overcoming these two challenges. Some reasons that CFD struggles to gain acceptance in this specific role include the following.

- Simulations using the Reynolds-Averaged Navier-Stokes equations are sometimes inadequate depending on the design of the vehicle, but higher-fidelity models are still too costly due to the large number of simulations required.
- Even where CFD is fully capable, it is easy for inexperienced teams to do it poorly, and relying on this data requires extra review.
- Many institutions have a successful legacy of a different way, and justifying a change is an extra hurdle—especially since wind tunnel testing would still be needed to estimate uncertainty, eliminating much of the potential cost saving.

B. Database Expansions

Two general additions to the databases from Section II are worth considering:

- include relevant data from all applicable CFD solutions, even for conditions outside the intended launch vehicle flight envelope, and
- implement interpolation for all databases.

For example, the surface pressure database should really include the skin friction coefficient vector for each point on the surface. In addition, existing databases for both the surface pressure and protuberance air load databases have been limited to 4° total angle of attack for the transonic portion of flight. This is true even though CFD simulations out to 8° are available, which are used in the integrated force & moment database uncertainty quantification. When the program later wants to decide if 4.25° is acceptable, significant challenges arise from not having all the necessary data already delivered to users of each database. Simply delivering extra data is not as trivial as it may sound in a large program, especially if the data in question is multiple years old.

Secondly, users should have the ability to request the value of any database value at any value of Mach number, angle of attack, and sideslip angle (i.e. M, α, β). In the SLS program, the integrated force & moment database and line load database already have this property, but the surface pressure and protuberance air load databases do not. Since the SLS run matrix (Section II.C) is designed to make this interpolation viable and accurate, this represents a missed opportunity. While interpolating the surface pressure of up to 8 million surface points may seem intimidating initially, the interpolated surface pressure field is nothing more than a linear combination of the existing CFD solutions. The main challenge for interpolating CFD surface pressure databases is delivering the interpolation scheme without causing undue difficulty on the user side; Section IV.B discusses one technique to address this problem.

C. Uncertainty Quantification for Distributed Aerodynamics

An inherent difficulty in the launch vehicle aerodynamics discipline, which arises from using different data sources for different databases, is that the distributed aerodynamic loads do not integrate to match up with the actual integrated loads. The actual challenge is even more difficult because even after adjusting the distributed loads, they would still not match the integrated loads that are dispersed according to the integrated force & moment uncertainty quantification. In [16], the authors discuss a solution for line load databases, and in [17] the authors introduce a solution for surface pressures.

While the details are left to those references, the basic concept is that a Proper Orthogonal Decomposition (POD) [18] is computed for either the surface pressure or the line loads of all CFD solutions at a constant Mach number. The resulting mode shapes are selected as candidates to add to the nominal CFD loads, and a constrained optimization problem is solved to create six modes for each Mach number. The first of these modes will increase the axial force C_A by 1.0 while leaving the other forces & moments unchanged, the second adjusts C_Y, etc. Figure 17 shows three of the force adjustment modes from two SLS Block 1 databases (one computed using FUN3D and one using OVERFLOW) for surface pressures at Mach 0.95. Users can thus add any linear combination of these six modes to match a target sextet of force & moment coefficients on the fly. While this technique has certain shortcomings and is not a true uncertainty quantification technique, it is designed with user simplicity in mind.

IV. The CAPE Software Package

The Computational Aerosciences Productivity & Execution (CAPE) software package[i] is an open-source[j] Python package developed by the authors of this paper to create common standards for running large run matrices of different CFD solvers. CAPE also provides tools that enable complex databases from CFD results without needing to develop substantial new software for each engineering problem. In version 1.0, CAPE has a common framework for running Cart3D, FUN3D, and OVERFLOW. Support for other solvers is in development. The CAPE package contains two major portions: one for preparing, running, extracting data, and archiving cases; and another portion for cleaning data and producing final data products. The first of these is usually activated through executables `pycart`, `pyfun`, or `pyover`, and is described in Section IV.A. The second introduces software named "datakit" which combines the data and the reference implementation into a single file as a Python package and is described further in Section IV.B.

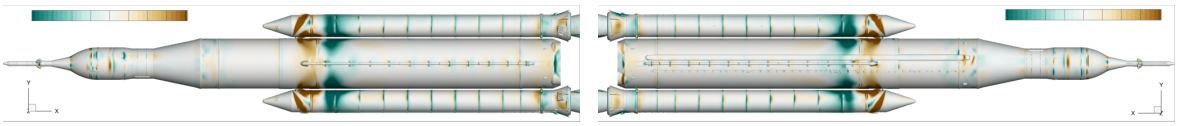
[i] https://software.nasa.gov/software/ARC-18753-1
[j] https://github.com/nasa/cape

A. CAPE CFD Run Executives

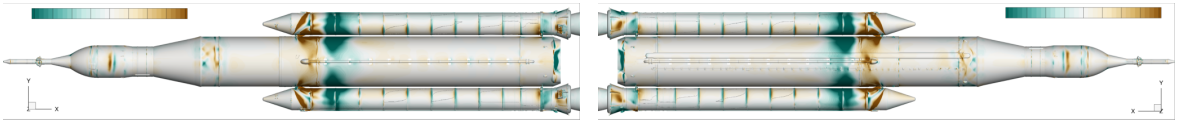
The CAPE package has three primary executables: `pycart` for Cart3D, `pyfun` for FUN3D, and `pyover` or OVERFLOW. Documentation for these is extensive[k] and has numerous examples[l]. However, there are some aspects particular to ascent CFD work for SLS that are interesting to discuss here. In particular, in addition to the properties listed in the Section I introduction, the SLS team runs these ascent run matrices as a group, in which multiple users are running cases. A launch vehicle ascent CFD database as large as SLS requires an amount of compute resources and individual work that splitting up the run matrix is advantageous, but it introduces complications like ensuring no cases get run by two different users and avoiding overwriting post-processed data. CAPE has some special capabilities for shared run matrices like this, and the SLS aerodynamics team has adopted a strategy using the version control software `git`[m] that has widespread applicability.

A color-coded diagram of this git-based strategy is shown in Fig. 18. The diagram is divided into three "layers," which are represented as colored dashed-line boxes, and represent locations where each group of files exists. The working layer, shown as a maroon dashed box in Fig. 18, is where the main CFD work is done. The working layer is (a) located on a high-performance computing system, (b) where the CFD input files live, (c) the location from which the main CAPE commands are issued, and (d) where the CFD results are found.

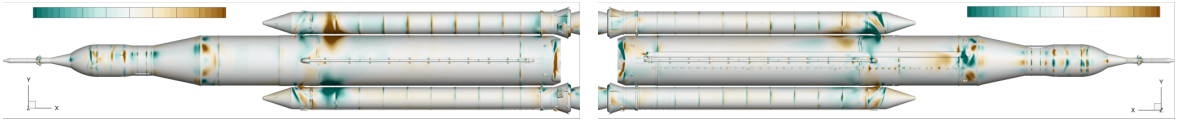
[k] https://nasa.github.io/cape-doc/
[l] https://github.com/nasa-ddalle
[m] https://git-scm.com/



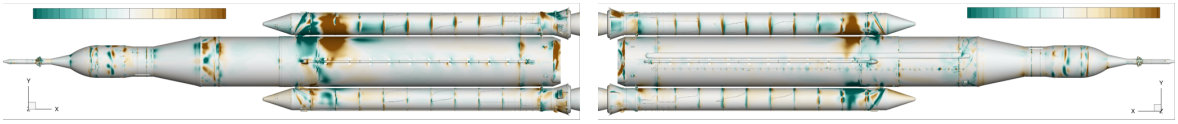
a) Mach 0.95 C_A +1 mode, FUN3D



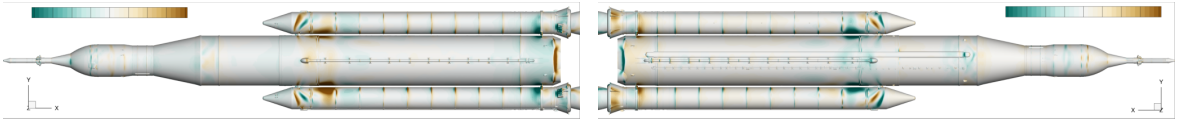
b) Mach 0.95 C_A +1 mode, OVERFLOW



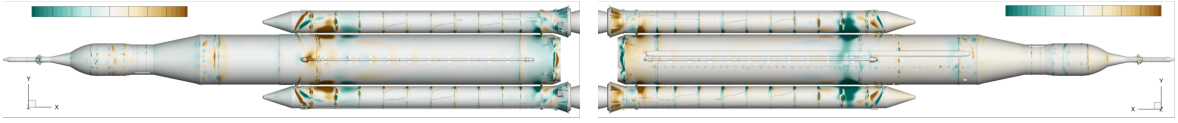
c) Mach 0.95 C_Y +1 mode, FUN3D



d) Mach 0.95 C_Y +1 mode, OVERFLOW



e) Mach 0.95 C_N +1 mode, FUN3D



f) Mach 0.95 C_N +1 mode, OVERFLOW

Figure 17. Force coefficient adjustment surface Δc_p profiles for Mach 0.95

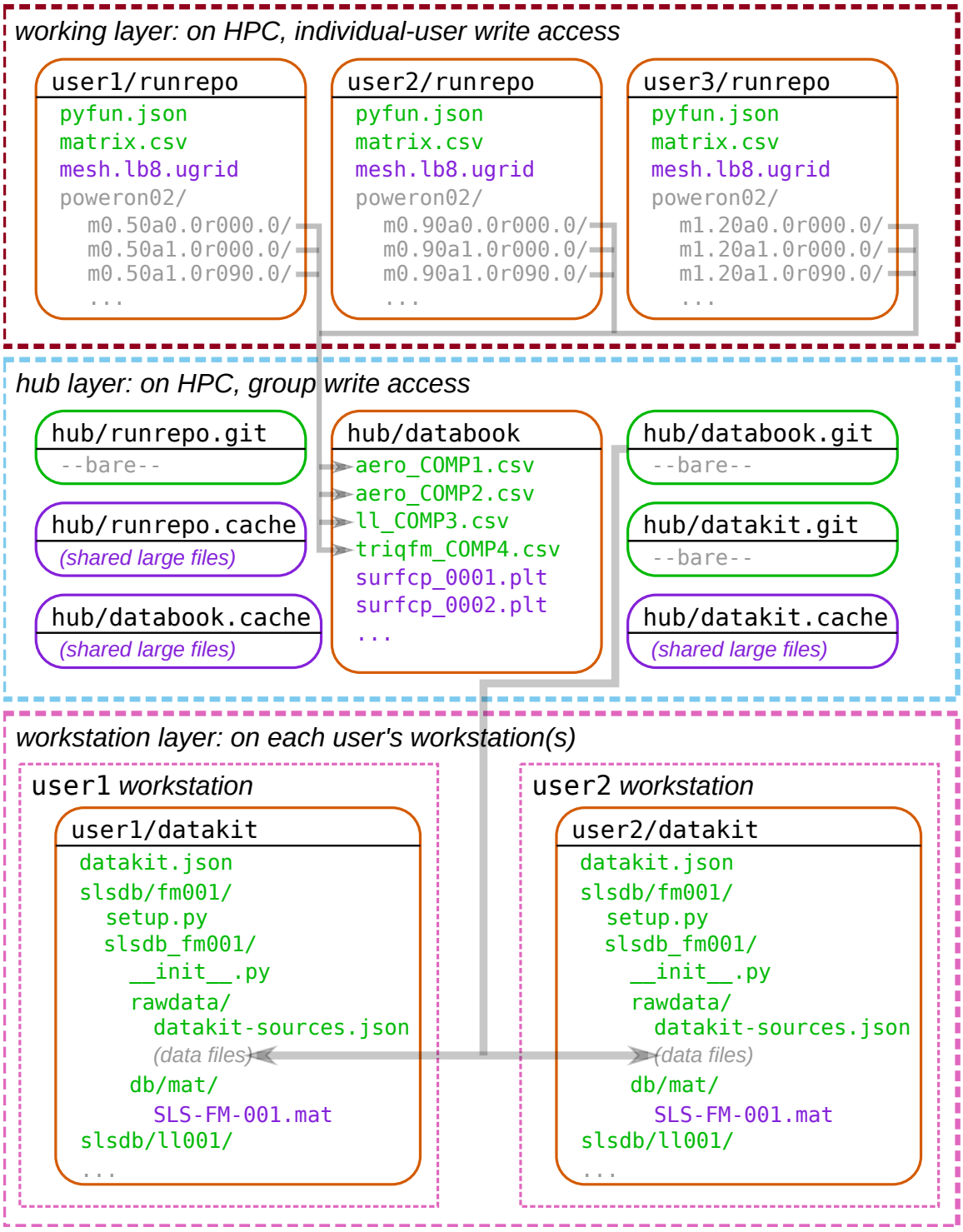


Figure 18. Diagram of repositories, large file caches, and data flow for a sample CAPE-based shared CFD run matrix

In the Fig. 18 example, three different users are running cases, and each has a separate version of the main repository for running cases. To make file sharing more efficient, ensure consistency, and preserve reproducibility, the CFD input template files and CAPE input files are tracked in a git repository. In Fig. 18, each orange box is a git repository ("repo" for short), and more specifically a working repository. A working repo is one that has current versions of each file and is where the majority of work happens. The working repos in Fig. 18 list some of the key files with color coding: green files are those tracked by git, purple files are tracked by a separate large file client[n], and gray files are not tracked at all. The actual CFD solutions, which are shown in gray, are not tracked by revision control software at all in this SLS team template for CFD using CAPE. As shown in the Fig. 18 example, each user maintains a separate run repo, which contains a subset of the CFD cases in the full run matrix. Furthermore, each individual user is the only team member with direct write access to their own run repo, although the entire group should have the ability to read the files.

The next layer, highlighted with a light blue dashed-line box in Fig. 18, is called the "hub layer"—a name inspired by the popular GitHub[o] tool. The hub layer is also located on the same high-performance computing system as the working layer, but each team member can read and write files here. Users share information about a git repo by pushing and pulling to a bare repo located in the hub layer. A bare repo is the opposite of a working repo; bare repos do not have current versions of each file but do contain the information to reconstruct each file and the entire history of each file. In this system, the large input files (for example the initial volume mesh) are shared by communicating with a large file cache also located in the hub layer, and these are shown as purple boxes in Fig. 18.

A novel aspect of the system described in Fig. 18 is that the post-processed data is located in the hub layer. Data extracted from CFD results using CAPE's post-processing commands are recorded in a separate location, and the collection of these CAPE post-processed data files is called the databook for a run matrix. For example the file `aero_COMP1.csv` contains the extracted integrated force & moment on a component called `COMP1` for every case in the run matrix, and the gray arrows in the top half of Fig. 18 attempt to demonstrate this. Having the working repo of the databook (orange box in the hub layer) is a key solution to challenges caused by splitting up a run matrix among team members. If the databook files are tracked as part of the run repo, then each user would have separate versions of each databook file, which causes frequent issues with merging these separate file versions. Having a single version of the databook, to which each user has write access, alleviates merge issues. CAPE has a built-in file locking system for the databook to avoid loss of data if multiple users try to write to them simultaneously.

The final layer in Fig. 18, shown as a pink dashed-line box, is called the workstation layer, which contains files and repos that users work on separate workstation systems outside of the high-performance computing system. Each user presumably has a separate workstation (shown as thinner pink dashed-line boxes), so users cannot communicate directly with each other in this layer; communication is done via push & pull commands to the bare repos in the hub layer. The repos that are shown in the workstation layer of Fig. 18 are the so-called datakits, which are the subject of Section IV.B. The important aspect for this description is that datakits take raw data from the databook as an input and develop that raw data (by removing bad data, mirroring, interpolating, etc.) into a final data set. In a datakit system, the databook files are usually considered raw data, and the gray arrows in the lower half of Fig. 18 demonstrate this data flow. CAPE has tools to automate this transfer so that a datakit (final data product) is simple to update without having to manually transfer databook files.

Regarding the actual CAPE commands used by the SLS team to run and process ascent CFD, the software documentation is a more appropriate source. To include a summary and tie it to a real application, the following paragraphs describe the commands used to take case number 830 from a FUN3D ascent CFD run matrix from beginning to end, through its entire life cycle using the CAPE infrastructure. A primary point of this example is to demonstrate CAPE's approach to use a single tool to manage as much of the CFD work as possible.

1. Submit the case

   ```
   pyfun -I 830
   ```

   This command copies necessary files, modifies templates, and submits a job to the HPC scheduler.

2. Generate a report

   ```
   pyfun -I 880 --report
   ```

[n]The precise nature of this tool for large file tracking is under revision; it may be included in a later version of CAPE
[o]https://github.com

This command generates a PDF report that can be customized by the user in the main CAPE input file. The main purpose of this report is to assure that a case ran correctly and is complete, but it often also serves as a database of common figures for a run matrix.

3. Extend the case (if needed)

   ```
   pyfun -I 880 --extend --qsub
   ```

   If the case ran its nominal sequence but appears to need more iterations, this command does that.

4. Repeat steps 2 and 3 as needed

5. Extract integrated force & moment data

   ```
   pyfun -I 880 --fm
   ```

   This command extracts averaged integrated forces & moments (possibly of many separate surface subset components) from the results in a case (in this example the case with index 880) and writes the results to file(s) (one file for each component) in the databook.

6. Extract other CAPE-supported data

   ```
   pyfun -I 880 --ll
   pyfun -I 880 --triqfm
   ```

   For ascent CFD run matrices, the first of these commands calculates and extracts line loads (including protuberance line loads), and the second extracts protuberance integrated loads.

7. Extract custom data

   ```
   ./tools/writesurfcp.py -I 880
   ```

   Some data may be important to users but not directly implemented by CAPE. This includes the surface pressure database, but even in these cases users can gain significant advantages by using the CAPE programmable interface in Python.

8. Archive the case

   ```
   pyfun -I 880 --archive
   ```

   This step, which is also highly customizable in CAPE, copies important files to a data backup server (in many cases packaging them into groups of related files) and deletes some large files in the working layer. By convention, the SLS team usually configures this command so that the data extraction is still possible after a `--archive` command has been run.

9. (*optional*) Remove even more files

   ```
   pyfun -I 880 --skeleton
   ```

   CAPE contains a second layer of archiving, which is customizable but by convention leaves only some basic or key information about a case. This first performs the archiving action, so no data is lost permanently.

10. (*optional*) Revive case from archive

    ```
    pyfun -I 880 --unarchive
    ```

    In some occasions, sometimes years after a run matrix is complete, someone from a program may request information about a case that was not extracted during the original run. The `--unarchive` command revives a case from the archive so that all of the original files are restored. If the software used for the original run is still supported, users should even be able to run more iterations following this command. One of the reasons this command is implemented in CAPE is to give users the confidence to delete files from the working file systems, where space is often at a premium.

B. DataKit Tool for Aeroscience Data Sets

A major portion of CAPE is a package called "datakit" for processing raw data and producing polished databases. Many of the databases that are the final products of complex computational aerosciences efforts produce a large dataset and a set of instructions to interpolate between actual CFD conditions. The idea of datakit is to combine the pure data of a traditional database and a toolkit to implement the verbal or written descriptions of how to use it. In most cases the final product of a computational aerosciences effort for a large engineering program is a database. However, unlike most databases, users of databases developed from CFD solutions rarely just need to lookup values from the exact conditions at which CFD was run. For example, users of an ascent CFD line load database, discussed in Section II.E, the SLS loads team needs to know the distributed aerodynamic loads not just at the points in a CFD run matrix but any arbitrary combination of Mach number, angle of attack, and sideslip angle.

The primary purpose of a datakit is to combine the data and this interpolation implementation into a single file using a generic framework that can successfully describe most data sets and interpolation/lookup methods. Not all users will be able to use this implementation directly, for example due to compatibility limitations between different programming languages, but even then it provides a useful reference implementation that can be tested extensively.

To classify the different levels of adoption of the CAPE datakit package that are possible, the developers have created four tiers, described below.

1. Simple datakit: usually one or more data files and one Python file
2. Datakit package: a full implementation of one database and response method packaged into a Python wheel[p].
3. Datakit collection: two or more datakit packages combined into a single repository
4. Datakit ecosystem: multiple datakit collections with tools to make the data universally accessible

1. *Simple Datakits*

An entry-level, tier 1 datakit can be as simple as a single data file and a single Python file that reads it and implements one or more lookup techniques. For example it might have a file named `aero_STACK_no_base.csv` from an ascent CFD databook with the forces and moments on the full SLS stack excluding the base and a file `datakit.py` that contains the following code after a few lines of configuration:

```
db = DataKit("aero_STACK_no_base.csv"}
db.make_responses(["CA","CY","CN"], "nearest", ["mach","alpha","beta"])
```

With this setup, `db("CN", 1.23, 4.1, -0.2)` returns the CFD result for the normal force coefficient, *CN* nearest to Mach 1.23 (`"mach"`), 4.1° angle of attack (`"alpha"`), and −0.2° sideslip angle (`"beta"`). Naturally, nearest-neighbor interpolation is not the most desirable technique; a more common implementation would be to first interpolate the CFD results onto a regular grid of Mach, α, and β values and then instruct the datakit to use trilinear interpolation. This can be done with a datakit tool like `db.regularize_by_rbf()` and then replacing `"nearest"` with `"linear"` in the `db.make_responses()` call.

2. *Datakit Packages*

The simple tier 1 datakit implementation can successfully achieve the goal of combining "database" and "toolkit" as the name suggests, but it does not have a straightforward method to combining all the data and a reference implementation into a single file. A tier 2 datakit implementation, or datakit package, is not much more challenging to create than tier 1, but it requires several separate files rather than the single `.py` file from tier 1. In order to create these files, CAPE includes a command-line executable called `dkit-quickstart` that creates the basic template. This command takes a single argument, which is the name of the datakit package that the user wishes to create. For example, to create a datakit package called `mydb001`, run

```
dkit-quickstart mydb001
```

This command first prompts the user for a plain-text title (for metadata purposes) and then notifies the user which files it creates. Details of each file created need not be discussed here, but as a general overview, the file layout conforms to the standard Python packaging tool called `setuptools`[q]. The standard for datakit packages is to consider a file like

[p] https://pythonwheels.com/
[q] https://pypi.org/project/setuptools/

`aero_STACK_no_base.csv` to be "raw data," and so by convention it should go into the `mydb001/rawdata/` folder. The example `datakit.py` file discussed in the tier 1 example now becomes `mydb001/__init__.py`, although it should be modified to fit the outline created by `dkit-quickstart`. For a datakit package, the standard is to read raw data and then write the cleaned data to a separate file. To do this, execute

```
dkit-writedb mydb001
```

which creates the file `mydb001/db/mat/datakit.mat`. A common scenario is that this file has regularized data, so it differs in contents from the raw data, and then the datakit can load its data quickly without performing additional operations after this initial `dkit-writedb` command. To create that one-file version of the datakit package, run

```
python setup.py build
python setup.py bdist_wheel
```

This will create a file with a name something like `mydb001-1.0-py3-none-any.whl`, and then other users can take delivery of the data by installing it:

```
python -m pip install --user mydb001-1.0-py3-none-any.whl
```

Datakit packages have many other capabilities, and a couple of them warrant a brief introduction here:

- The `rawdata/` folder can be set up to automatically retrieve data from a remote repository. That way if a databook or other data source is updated, the datakit is easy to update (without manual file copying) as well.
- Specific versions of third-party or internally developed Python packages can be "vendorized" within the datakit. This enables creators of datakit packages to use early-development Python tools and freeze them so that the datakit is safe from any future software updates that might break that datakit.

3. Datakit Collections

A datakit collection is simply a folder (usually a git repository) containing multiple datakit packages. This may seem like a rather trivial concept to receive its own name and description, but it enables some important features that are essential to the overall datakit tools. Most importantly, a datakit package can easily be made to rely on other datakit packages within the same collection. For example, a collection could include separate datakits:

1. raw forces & moments from CFD of the flight-scale vehicle,
2. raw forces & moments from CFD of a wind tunnel test,
3. regularized tables of the flight-scale CFD,
4. regularized tables of the wind-tunnel CFD,
5. tables of the difference between the two at all conditions, and/or
6. an uncertainty quantification based on those differences.

In this example, item 3 depends on 1, 4 depends on 2, 5 depends on 3 & 4, and 6 depends on 5. A datakit collection makes it easy to create a pipeline so that updates in the raw data (for example by running a few additional CFD cases) can be quickly processed to update the other datakits that depend on it.

Another important aspect of the datakit collection design is that it enables collaboration of team members on data in a novel way. For example in the SLS ascent CFD group, all the Block 1 force & moment database implementations are found in a single repo called `SLS-10-D-AFA`. This gives the team access to all the most recent datakits easily and provides a framework to make data processing more like software development.

4. Datakit Ecosystems

Much like the relationship between tier 2 and tier 3, a datakit ecosystem is a group of one or more datakit collections. The main characteristic of the datakit ecosystem is that it enables team members to easily read any datakit in the entire list of datakit collections using a simple command. To do this, CAPE has a module called `datakithub`, providing a class of the similar name `DataKitHub`. Once a hub object is instantiated, any datakit that has been incorporated into the ecosystem can be read by name. For example, the official ascent force & moment database in service for the successful launch of Artemis I is called `SLS-10-D-AFA-004`. Reading this database using the datakit ecosystem used by the SLS ascent CFD team looks like the following:

```
hub = DataKitHub()
db = hub.read_db("SLS-10-D-AFA-004")
```

Once established, this approach greatly reduces the often time-consuming effort to locate data files and code, To get to this point, users need to set up a few files and have a good understanding of Python regular expressions[r] to help the `DataKitHub` find the data and convert a database name like `SLS-10-D-AFA` to a module name like `sls10afa.v004.rev0`. Once these patterns are established, the datakit ecosystem can be a major time saver and automate the transition of data from the original developer to an entire team.

V. Future Advances and Long-Term Vision

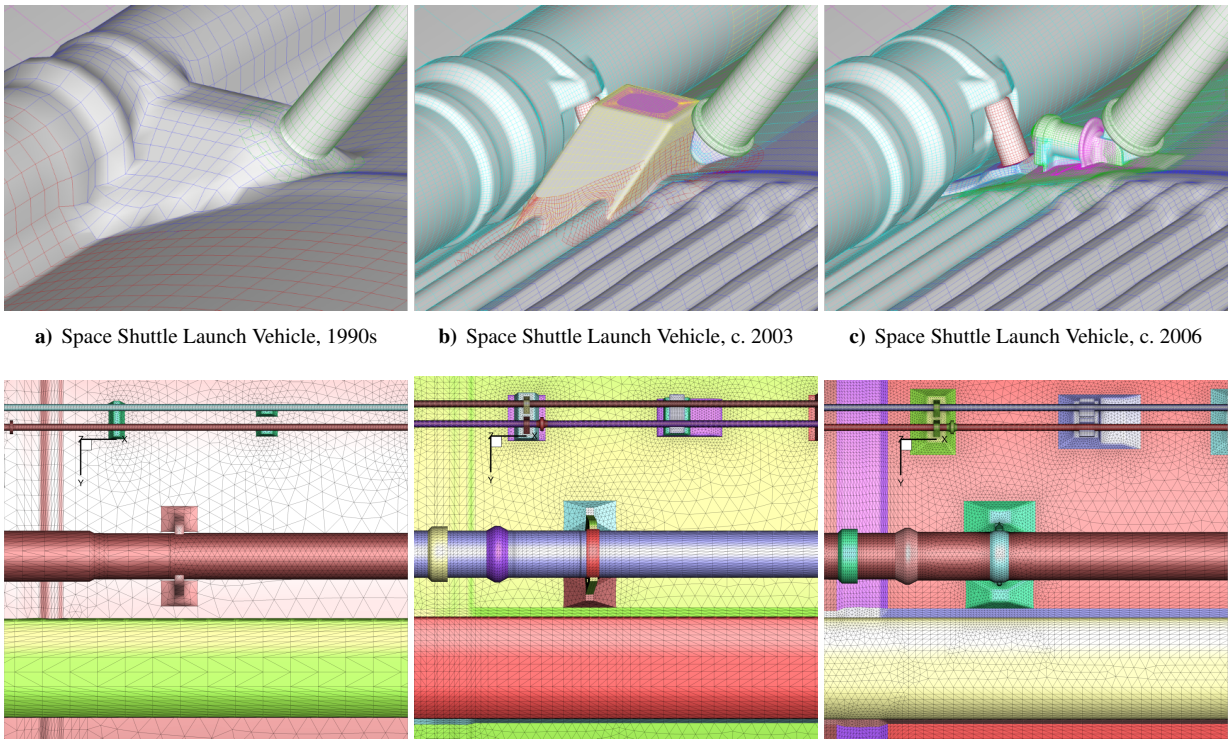
Broadly, the long-term vision for ascent CFD is to be used and trusted for as much of launch vehicle aerosciences as possible. This statement does not imply a future preference for computational results over wind tunnel testing, but coming expansions to computing capacity and numerical algorithm improvements will expand the areas where CFD is validated and practical. Many areas for potential improvement remain, and additional capabilities range from just around the corner to tasks that are still years away. Three areas of improvement in the vision for launch vehicle ascent CFD are *consistency*, *capacity*, and *uncertainty*.

Consistency improvements aim to address discrepancies between different data products and how they are used, and this includes creating consistency after uncertainty dispersion. For example integrating the distributed loads from surface pressure and line load databases should match up with the integrated force & moment database. Another type of consistency is that more products could come from the same CFD simulations, for example base pressure and aerothermal analysis, which are currently modeled using separate CFD simulations with different assumptions. Capacity improvements describe things that would be possible with expanded computing capacity and improved numerical efficiency. At the current stage, every expansion of available high-performance computing leads directly to improved launch vehicle ascent CFD simulations. Attaining the long-term vision of gathering steady loads and unsteady aerodynamics, including both buffet and acoustics, from the same computational simulations requires a substantial increase in compute capacity and likely also improvements in efficiency since currently the two types of simulations are carried out with significantly different procedures. Increases in available computing resources will also improve the third category, which is uncertainty quantification. Techniques for directly estimating some aspects of uncertainty from CFD are becoming available and gaining acceptance [19], but they typically require many dedicated CFD solutions for each flight condition at which uncertainty is estimated. Directly estimating numerical uncertainty for a full database could easily multiply the overall computational requirements by 30 times because current techniques require roughly this number of separate solutions to estimate the uncertainty at one condition. Important aspects of the long-term vision for launch vehicle ascent CFD include the following.

1. Use of scale-resolving simulations instead of Reynolds-averaged Navier-Stokes for the entire database.
2. Numerical simulation of buffet loading (low-frequency aerodynamic loading) from same simulations as steady loads.
3. High-resolution unsteady loads available at many flight conditions rather than a few.
4. Direct estimation of unsteady protuberance.
5. Development of uncertainty quantification for line loads and surface pressures that are consistent with the integrated force & moment uncertainty.
6. Improved collaboration with downstream users of aerodynamic data so that more teams use case-consistent analysis and Monte-Carlo simulations to reduce conservatism.
7. Direct estimates of the consequences of geometric uncertainty.
8. Include additional physics in ascent CFD simulations so that base environment and aerothermal analysis can come from one set of simulations.

Some of these aspects will likely to require many years to realize. By that time, the nature of new space launch vehicles may be different enough that this long-term vision may change, too. Furthermore, due to the goal of combining some simulations that are currently carried out by separate teams, the amount of engineers required to do the work

[r] https://docs.python.org/3/library/re.html



a) Space Shuttle Launch Vehicle, 1990s **b)** Space Shuttle Launch Vehicle, c. 2003 **c)** Space Shuttle Launch Vehicle, c. 2006

d) Space Launch System, 2014 **e)** Space Launch System, 2017 **f)** Space Launch System, 2022

Figure 19. Historical NASA launch vehicle surface grid snapshots

may reduce. Rather than reducing labor, this hopefully will allow institutions to propose and develop new vehicles at a faster rate.

It is important to remember how far launch vehicle computational aerosciences has come within a single engineering career. Figure 19 shows a set of images of surface grids from NASA launch vehicles from various points in the last 30 years. In addition to the considerable improvement in grid resolution and geometric complexity apparent from Fig. 19, simulations are now performed at many more flight conditions. In the 1980s, such a simulation was not considered possible at all [1]; today it is almost commonplace, at least for large institutions. The advances of the last 30 years in computational aerosciences have in part been driven by the needs of launch vehicle programs. This trend is expected to continue. With this view of the state of launch vehicle ascent CFD over the last 30 years, the expectation is that the long-term vision outlined above is a dramatic underestimate of what will be possible in the next 30 years.

Acknowledgments

Resources supporting this work were provided by the NASA High-End Computing Capability (HECC) program through the NASA Advanced Supercomputing (NAS) Division at Ames Research Center. This work was funded by the Space Launch System (SLS) Program, part of the Artemis Program and the Exploration Systems Development Mission Directorate (ESDMD). The authors would like to thank Oliver Browne and James Koch for internal review.

References

[1] Martin, F. W. and Slotnick, J. P., "Flow Computations for the Space Shuttle in Ascent Mode Using Thin-Layer Navier-Stokes Equations," *Applied Computational Aerodynamics*, edited by P. Henne, chap. 24, American Institute of Aeronautics and Astronautics, 1990, pp. 863–886.

[2] Kiris, C. C., Barad, M. F., Housman, J. A., Sozer, E., Brehm, C., and Moini-Yekta, S., "The LAVA Computational Fluid Dynamics Solver," *52nd Aerospace Sciences Meeting*, 2014, AIAA Paper 2014-0070.

[3] Buning, P. G., Chiu, I. T., Obayashi, S., Rizk, Y. M., and Steger, J. L., "Numerical Simulation of the Integrated Space Shuttle Vehicle in Ascent," *15th Atmospheric Flight Mechanics Conference*, 1988, AIAA Paper 1988-4359.

[4] Slotnick, J. P., Kandula, M., and Buning, P. G., "Navier-Stokes Simulation of the Space Shuttle Launch Vehicle Flight Transonic Flowfield Using a Large Scale Chimera Grid System," *12th Applied Aerodynamics Conference*, 1994, AIAA Paper 1994-1860.

[5] Gomez, R., Vicker, D., Rogers, S., Aftosmis, M., Chan, W., Meakin, R., and Murman, S., "STS-107 Investigation Ascent CFD Support," *34th AIAA Fluid Dynamics Conference and Exhibit*, 2004, AIAA Paper 2004-2226.

[6] Koch, J. R., Dumlupinar, E., Housman, J. A., Kiris, C. C., Patel, M. M., Kleb, B., Brauckmann, G. J., and Alter, S. J., "CFD Analysis of Space Launch System Solid Rocket Booster Separation within the Langley Unitary Plan Wind Tunnel," *AIAA Aviation 2021 Forum*, 2021, AIAA Paper 2021-2966.

[7] Murman, S. M. and Diosady, L., "Simulation of a Hammerhead Payload Fairing in the Transonic Regime," *54th AIAA Aerospace Sciences Meeting*, 2016, AIAA Paper 2016-1548.

[8] Rogers, S. E., Dalle, D. J., and Chan, W. M., "CFD Simulations of the Space Launch System Ascent Aerodynamics and Booster Separation," *53rd AIAA Aerospace Sciences Meeting*, January 2015, AIAA Paper 2015-0778.

[9] Biedron, R. T., Carlson, J.-R., Derlaga, J. M., Gnoffo, P. A., Hammond, D. P., Jacobson, K. E., Jones, W. T., Kleb, B., Lee-Raush, E. M., Nielson, E. J., Park, M. A., Rumsey, C. L., Thomas, J. L., Thompson, K. B., Walden, A. C., Wang, L., and Wood, W. A., "FUN3D Manual: 13.7," Tech. Rep. TM-20205010139, NASA, 2020.

[10] Nichols, R. H., Tramel, R. W., and Buning, P. G., "Solver and Turbulence Model Upgrades to OVERFLOW 2 for Unsteady and High-Speed Applications," *24th Applied Aerodynamics Conference*, 2006, AIAA Paper 2006-2824.

[11] Biedron, R. T., Carlson, J., Derlaga, J. M., Gnoffo, P. A., Hammond, D. P., Jones, W. T., Lee-Rausch, E. M., Nielson, E. J., Park, M. A., Rumsey, C. L., Thomas, J. L., and Wood, W. A., "FUN3D Manual: 13.1," Tech. Rep. TM-2017-219580, NASA, 2017.

[12] Spalart, P. R. and Allmaras, S. R., "One-Equation Turbulence Model for Aerodynamic Flows," *La Recherche Aerospatiale*, Vol. 1, No. 1, 1994, pp. 5–21.

[13] Buning, P. G. and Pulliam, T. H., "Cartesian Off-Body Grid Adaption for Viscous Time-Accurate Flow Simulations," *20th AIAA Computational Fluid Dynamics Conference*, 2011, AIAA Paper 2011-3693.

[14] Rogers, S. E., Roth, K., Nash, S. M., Baker, M. D., P., S. J., Whitlock, M., and Cao, H. V., "Advances in Overset CFD Processes Applied to Subsonic High-Lift Aircraft," *18th AIAA Applied Aerodynamics Conference*, June 2000, AIAA Paper 2000-4216.

[15] Pandya, S. and Chan, W. M., "Computation of Sectional Loads from Surface Triangulation and Flow Data," *20th AIAA Computational Fluid Dynamics Conference*, 2011, AIAA Paper 2011-3680.

[16] Dalle, D., Rogers, S., and Meeroff, H. L. J., "Adjustments and Uncertainty Quantification for SLS Aerodynamic Sectional Loads," *2018 Applied Aerodynamics Conference*, 2018, AIAA Paper 2018-3640.

[17] Dalle, D. J., Rogers, S. E., Burkhead, A. C., and Meeroff, J. G., "Distribution of SLS Integrated Load Uncertainty to Surface Pressures and Sectional Loads," *Eleventh International Conference on Computational Fluid Dynamics*, 2022, ICCFD11-3303.

[18] Hall, K. C., Thomas, J. P., and Dowell, E. H., "Proper Orthogonal Decomposition Technique for Transonic Unsteady Aerodynamic Flows," *AIAA Journal*, Vol. 38, No. 10, 2000, pp. 1853–1862.

[19] Barth, T., "An Overview of Combined Uncertainty and A-Posteriori Error Bound Estimates for CFD Calculations," *54th AIAA Aerospace Sciences Meeting*, 2016, AIAA Paper 2016-1062.