Parametric Domain Representation of Multi-Block Structured Meshes for Strong-Shock Flow Simulations

Ryan Gosse[1] Cameron Mackintosh[2], Konstantinos Vogiatzis[3], Charles Williams[4]
U.S. Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio, 45433

Procedures to parametrically represent two- and three-dimensional domains, used in generation of structured meshes for CFD (Computational Fluid Dynamics) simulations, with Bezier surfaces and volumes have been developed. They provide compact data structures, exhibiting several orders of magnitude reduction in size for the same fidelity that can facilitate scalable mesh generation, decomposition and data analysis and eliminate bandwidth saturation. A demonstration of multi-block mesh generation using an example shape pertinent to compressible strong-shock flow simulations is presented.

I. Introduction

Simulations that are approaching the Exascale in High Performace Computing (HPC) for external aerodynamics require mesh sizes that are in the tens to hundreds of billions of elements. The challenge that this imposes is many-fold. Commercial mesh generation software exhibits limitations in parallel fucntionality, maximum memory and element count, and file I/O speeds. Especially the latter is often the bottleneck in the process of mesh generation, partitioning and loading to the CFD solver, consuming the HPC system I/O bandwidth and adversely impacting other users. Another aspect of multi-billion element meshes is the daunting disk space requirement just for storage of the discrete system.

To this end a procedure and code to parametrically represent two- and three-dimensional domains, used in generation of structured meshes for CFD simulations, with Bezier parametric definitions have been developed. Bezier representation of curves and surfaces are standard parametric definitions used in the Computer Aided Design (CAD) industry. Being an algebraic method, it is computationally inexpensive and trivial to parallelize. This makes it attractive for mesh sequencing, automatic mesh refinement and moving/overset mesh applications. The information contained in this representation is critical for defining higher order mesh elements to support the spatial discretization methods currently under development for pre-exascale HPC systems. In addition, model creation, manipulation and post-processing (mesh generation, domain decomposition, data analysis) on a single HPC node is becoming inefficient, if not impractical, for multi-billion element meshes. Techniques are needed to allow for those tasks to be executed in a scalable manner on HPC systems. Priority is placed upon algorithms that scale well on hybrid multi-core and distributed platforms. This approach will provide compact data structures, exhibiting several orders of magnitude reduction in size for the same fidelity that can facilitate scalable mesh generation, decomposition and data analysis and eliminate bandwidth saturation.

II. Methodology

A. Mathematical background

Bezier curves were named after Pierre Bezier, who first used them to design automobile bodies in 1962. P. de Casteljau also independently developed the definition in 1958 working for Citroen Car Company in France. They

[1] Senior Research Aerospace Engineer, Aerospace Systems Directorate, AFRL/RQHF, 2210 Eighth Street, Associate Fellow.
[2] Summer Intern, DoD HPCMP PETTT/Engility Corp., DoD Supercomputing Resource Center, AFRL/RCM, 2435 Fifth Street.
[3] Computational Scientist, DoD HPCMP PETTT/Engility Corp., DoD Supercomputing Resource Center, AFRL/RCM, 2435 Fifth Street, Senior Member.
[4] Computational Scientist, DoD HPCMP PETTT/Engility Corp., DoD Supercomputing Resource Center, AFRL/RCM, 2435 Fifth Street, Member.

arose from the need to store faired curve cross-sections used at the time to define drafting sketches. This was accomplished by storing the curves as a list of points. This allowed designs to have an exact mathematical definition instead of the imprecise mold and "master" approach used at the time. Bezier curves can be thought of as an extension of Berstein polynomials.

A Bezier curve is defined in entirety by its vector of control points; any point's real space coordinates $x, y, z \in \Re$ can be calculated from these control points. Points are represented by their computational space parameters $u, v, w \in [0,1]$. Similarly, a Bezier surface is defined entirely by its matrices of control points and they can be extended to n number of dimensions. In three dimensions the Berstein representation of a Bezier volume is described by,

$$V(u,v,w) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} \sum_{k=0}^{n_w} B_{n_u,i}(u) B_{n_v,j}(v) B_{n_w,k}(w) P_{i,j,k} \tag{1}$$

where $P_{i,j,k}$ are the family of control points defining the volume. The Bernstein weighting coefficients are defined by,

$$B_{a,b}(t) = \begin{pmatrix} a \\ b \end{pmatrix} t^b (1-t)^{a-b} \qquad b = \{0, \ldots, a\} \tag{2}$$

where,

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{a!}{b!(n-b)!} \tag{3}$$

is the binomial coefficient. The V and P vectors represent the real-space coordinates while u, v, w are the parameter-space coordinates defined in the interval [0,1]. The values n_u, n_v, n_w define the degree of the function in each u, v, w space direction. To represent a grid system three Bezier functions are used.

$$x(u,v,w) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} \sum_{k=0}^{n_w} B_{n_u,i}(u) B_{n_v,j}(v) B_{n_w,k}(w) P_{X\,i,j,k} \tag{4}$$

$$y(u,v,w) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} \sum_{k=0}^{n_w} B_{n_u,i}(u) B_{n_v,j}(v) B_{n_w,k}(w) P_{Y\,i,j,k} \tag{5}$$

$$z(u,v,w) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} \sum_{k=0}^{n_w} B_{n_u,i}(u) B_{n_v,j}(v) B_{n_w,k}(w) P_{Z\,i,j,k} \tag{6}$$

Note that the Bernstein coefficients are computed once and can be applied to all three equations. They can also be precomputed and stored for every u, v, w spacing definition. So when the grid "moves" due to control point translation, the only operation is a multiplication operation on a single vector of data. In addition, the Bezier definition is attractive due to:

1. The convex hull of its control points contains the parameter space. Hence a group of Bezier patches can be translated and stretched with out the "volume" of the function folding on itself so long as the control points of a polygon group do not cross.
2. The control points have a global weighting system that smoothly varies from 0 to 1 from end to end. This ensures that the control points of a Bezier volume on the surface simplify to a Bezier surface.
3. The derivative of the Bezier definition is trivial to compute at any location in space.
4. The continuity between patches is trival to preserve. Patch singularities can become problematic, but algorithms have been developed to address this issue.
5. Bezier curves never oscillate more than their control points.

B. **De Casteljau Algorithm**

Unfortunately the Bezier definition is problematic due to large values of the binomial coefficient for high degree functions causing computational complexity and numerical instability. De Casteljau, during his algorithm development, proposed an elegant solution that relies on the concept of sub-division based solely on the control points. For a Bezier curve the algorithm is as follows

$$P_i^k(t) = \begin{cases} (1-t)P_i^{k-1} + tP_{i+1}^{k-1}, & if \quad k > 0 \\ P_i \quad , & if \quad k = 0 \end{cases} \quad for \quad \begin{cases} i = 1,\ldots,n-k \\ k = 0,\ldots,n \end{cases} \tag{7}$$

where the Bezier control point is used at the k = 0 level of the loop and new mid-points are found. At the end of the k loop the final value $P_0^n(t)$ is the point on the curve. This algorithm can be extended to n parameter dimensions by taking advantage of the commutative properties of the Bezier definition. For example, in three parameter dimensions Eq. (1) can be processed by the de Casteljau algorithm giving,

$$V(u,v,w_o) = \sum_{i=0}^{n_u}\sum_{j=0}^{n_v} B_{n_u,i}(u)B_{n_v,j}(v)P_{i,j,0}^{\;n_k}(w_o) \tag{8}$$

where $P_{i,j,0}^{\;n_k}(w_o)$ is the new set of control points that define the surface slice at a fixed value of w = w$_o$. We can then apply this at a fixed value of v = v$_o$ giving:

$$V(u,v_o,w_o) = \sum_{i=0}^{n_u} B_{n_u,i}(u)P_{i,0,0}^{\;n_k,n_j}(v_o,w_o) \tag{9}$$

And finally for a fixed value of u = u$_o$.

$$V(u_o,v_o,w_o) = P_{0,0,0}^{\;n_k,n_j,n_i}(u_o,v_o,w_o) \tag{10}$$

This allows for a compact way to cast an i, j, k block system of grids, where saving the control points from each reduction can be applied at each new value of u for v$_o$ and w$_o$.

C. **Spatial Discretization of the Grid Domain**

One disadvantage of the Bezier definition is the fact that the function is evaluated in u, v, w parameter space. This does not directly map arc-length distance (for a Bezier curve) as one may expect. In order to create a spatial x, y, z grid, an additional mapping function must be defined. For the case of the Bezier curve this approximate mapping to the tolerance of the grid requirements is trivial by computing the arc-length of the curve

$$L_B = \int_0^t \sqrt{B_x'(t)^2 + B_y'(t)^2 + B_z'(t)^2}\,dt \tag{11}$$

and the derivative for a curve is found using:

$$B'(t) = \sum_{i=0}^{n-1} B_{n-1,i}(t)\left[n\left(\vec{X}_{i+1} - \vec{X}_i\right)\right] \tag{12}$$

Using the above equations to create an array of points along t, we can fit another Bezier curve to map required t space given a desired arc-length (grid point spacing distribution). A similar remapping can be used for Bezier surfaces and volumes to ensure strict control over grid point spacing. To solve this problem, we introduce arc length tables. The larger the initial mesh is, the better it approximates the intended surface, and thus the smaller the arc length parameterization error is. We exploit this triviality by generating a large intermediate mesh with a uniform computational-space distribution; we maintain a table mapping the generating computational-space values (those used to generate a point's real-space value) to calculate the actual values of each point by applying the definition of arc length. Equations (13) – (16) show its application to a Bezier surface and can be extended easily to a volume.

$$u_{actual_{i,j}} = \frac{L_u\left(u_{gen_{i,j}}, v_{gen_{i,j}}\right)}{L_u\left(1, v_{gen_{i,j}}\right)} \tag{13}$$

$$v_{actual_{i,j}} = \frac{L_u\left(u_{gen_{i,j}}, v_{gen_{i,j}}\right)}{L_u\left(u_{gen_{i,j}}, 1\right)} \tag{14}$$

$$L_u(u,v) = \int_0^u \sqrt{\left(\frac{\partial x(u,v)}{\partial u}\right)^2 + \left(\frac{\partial y(u,v)}{\partial u}\right)^2 + \left(\frac{\partial z(u,v)}{\partial u}\right)^2}\, du \tag{15}$$

$$L_v(u,v) = \int_0^v \sqrt{\left(\frac{\partial x(u,v)}{\partial v}\right)^2 + \left(\frac{\partial y(u,v)}{\partial v}\right)^2 + \left(\frac{\partial z(u,v)}{\partial v}\right)^2}\, dv \tag{16}$$

The partial derivatives can be calculated from the de Casteljau algorithm or Bezier equations. We can now generate new meshes such that the real-space distribution of the new mesh preserves the generating computational-space distribution. We binary search the arc length table's actual computational-space values for our desired generating computational-space values, linearly interpolating as needed. We create a new computational-space distribution, replacing our desired distribution's values with their corresponding actual values. We now can apply the de Casteljau algorithm with our new computational-space distribution to generate a new mesh.

A large (and thus computationally intense) arc length table is necessary to sufficiently decrease arc length parameterization error as the table is queried via binary search and linear interpolation. We can instead query the table by fitting a Bezier curve to it and applying the de Casteljau algorithm. This approach permits us to drastically reduce the size of the table and thus the computational expense.

D. Coordinate Transform of the Grid Domain

The previous section assumes that we are creating a grid for a generic solver that uses a discretized domain of x, y, z points to compute on. Instead we can apply the Bezier definition itself to aid in solving the solution domain. For a computational domain we often need to find the derivative of a function. For example, we look at a two dimensional problem where,

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial x} \tag{17}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial y} \tag{18}$$

are the derivatives of a value in the x and y direction and we apply a coordinate transform to the u, v space of the Bezier surface. The transformation derivatives are found by:

$$\frac{\partial u}{\partial x} = \frac{\dfrac{\partial y}{\partial v}}{\dfrac{\partial x}{\partial u}\dfrac{\partial y}{\partial v} - \dfrac{\partial y}{\partial u}\dfrac{\partial x}{\partial v}} \tag{19}$$

$$\frac{\partial v}{\partial x} = \frac{-\frac{\partial y}{\partial u}}{\frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial y}{\partial u}\frac{\partial x}{\partial v}} \tag{20}$$

$$\frac{\partial u}{\partial y} = \frac{-\frac{\partial x}{\partial v}}{\frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial y}{\partial u}\frac{\partial x}{\partial v}} \tag{21}$$

$$\frac{\partial v}{\partial y} = \frac{\frac{\partial x}{\partial u}}{\frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial y}{\partial u}\frac{\partial x}{\partial v}} \tag{22}$$

We can recast Eq. (12) to have the form,

$$B'(t) = \sum_{i=0}^{n-1} B_{n-1,i}(t)\overline{P}_i \tag{23}$$

Which looks exactly like the original Bezier definition and can apply the de Casteljau algorithm to compute the exact transformation derivatives for our discretization mapping stencils that are distributed in space t. Again note that for a moving Bezier grid system, the control points alone that change value. We can save the Bernstein coefficients for a fixed t space (grid point distribution mapping) and quickly compute new grid transformation metrics.

E. **Integrals of the parameterized space**

The integral over the entire Bezier definition is also trivial to compute. It is the barycentre of the control points. We will show the example of integrating a curve. Equation (24) defines the integration process.

$$\int_0^1 c(t)dt = \frac{1}{n+1}\sum_{i=0}^{n} P_i \tag{24}$$

Unfortunately, we would like to compute integrals over internal sections of the parameterized domain to compute things such as the surface area of a flux or volume of a source term. In order to integrate a subsection, we need to cut the Bezier curve into multiple children Bezier curves. This can be done again using the de Casteljau algorithm. Remember that the algorithm recursively sub-divides the Bezier curve until a control point lies on the curve itself. This control point is the location of two new Bezier curves touching at location t. During the de Casteljau iteration loop k if we collect the minimum and maximum i values (Eq. (7)), these values are the control points that define the two children Bezier curves. We can apply this method to multiple ranges of t_1 to t_2 and find control points that define a Bezier curve over any section of interest to integrate using Eq. (24). We can then use this process to cut out our local section of the grid to compute on. It is important to point out that this works for any structured data sets such as i, j, k blocks or Adaptive Mesh Refinement (AMR) grids where a u, v, w parameter mapping space can be constructed.

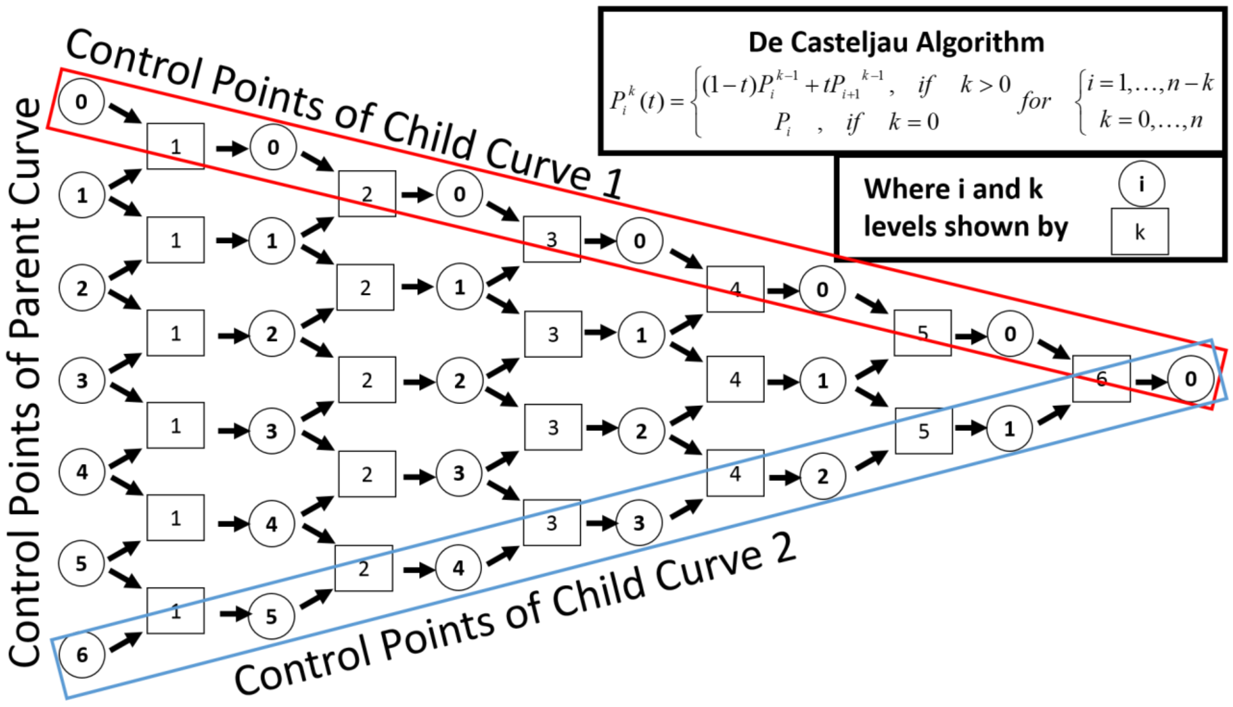
Control Points of Child Curve 1
Control Points of Parent Curve
Control Points of Child Curve 2
De Casteljau Algorithm
$P_i^k(t) = \begin{cases} (1-t)P_i^{k-1} + tP_{i+1}^{k-1}, & if \quad k > 0 \\ P_i, & if \quad k = 0 \end{cases} \; for \; \begin{cases} i = 1,\ldots,n-k \\ k = 0,\ldots,n \end{cases}$
Where i and k levels shown by
i
k

Figure 1. Visual sub-division of Bezier curve using de Casteljau algorithm. Note that the computed values along the outside of the triangle define the new control points of the two new children curves.

III. Grid Parameterization

A. Surface Parameterization

The CAD industry mostly focuses on curves and surfaces and there is a vast literature of how to apply the methods from the previous section to them. Therefore, we first applied our fitting algorithm to surfaces to test accuracy and speed. A small test program was written in python with the tested Bezier algorithms written in FORTRAN. We measured how the new meshes real-space distributions correspond to their computational-space distributions by generating our new meshes with uniform distributions and calculating the average segment length standard deviation of each row and column. We measure how the new mesh approximates the original geometry by calculating the normalized root-mean-square deviation (NRMSD) of the distance between each point's parameterized real-space location and its corresponding original geometry real-space location. The average segment length of the torus geometry normalizes the deviation. We measure a set of procedure computational complexity by averaging the user-space runtime on a specific machine and compiler setting (in this case we will average 100 executions on a DoD HPC Thunder compute node with Intel –O2 –Ofast optimizations).

Our first comparison was for a geometry of a torus section with major radius of 2.0 and minor radius of 1.0. We first generated a coarse mesh with one block and dimension 20 x 20. We parameterized it by approximated arc length. From this, we generated a fine mesh with one block, mesh dimensions of 100 x 100, fifth order Bezier surface, arc length table dimensions 50 x 50, and a uniform generating computational-space distribution. We calculated the discussed statistics for several combinations of the dicussed procedures shown in Table 1.

The de Casteljau algorithm is slightly faster and slightly more accurate than the Bezier equations in almost every case. The average standard deviation of the segment length dramatically decreases and the run-time slightly increases after implementing arc length tables. Querying the arc length table by Bezier fit is slightly slower and slightly more accurate than querying the table by linear interpolation. For CFD computations with viscous boundary layers, it is important to note that wall streatching functions are employed in the wall normal direction. They are typically a factor of 10 to 1000 times smaller than the segment length in the tangential direction. Therefore just using the Bezier representation without the arc length table can be problematic with average segment length standard deviation on the order of 1e-02. The initial coarse mesh and new fine mesh are pictured below.

We also verified our test code's computational time complexity, ensuring it matches the complexity of our procedures. With respect to the number of control points, the complexity is of O(n^2), as pictured below and as desired. With respect to the number of points in the new mesh, the complexity is of O(n) as pictured below and as desired.

Table 1. Results for fitting to a torus surface segment

	Average Segment Length Standard Deviation	NRMSD of Distance to Geometry	Average User-space Runtime (seconds)
Bezier equations and no arc length table	3.015E-02	5.936E-02	2.966E-03
De Casteljau algorithm and no arc length table	3.042E-02	5.798E-02	2.856E-03
Bezier equations and linear arc length table	5.684E-07	5.530E-02	3.115E-02
De Casteljau algorithm and linear arc length table	5.44E-07	5.222E-02	3.107E-02
Bezier equations and fitted arc length table	2.446E-10	5.747E-02	6.820E-02
De Casteljau algorithm and fitted arc length table	2.239E-10	5.589E-02	5.304E-02

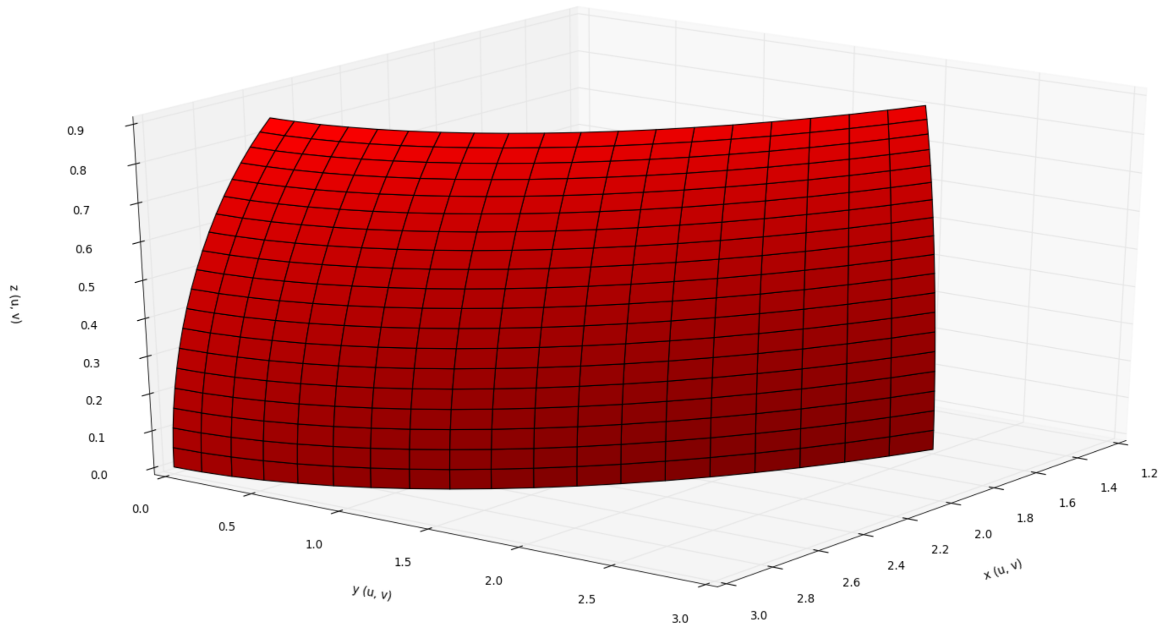

Figure 2. Example initial coarse mesh for a torus surface segment.

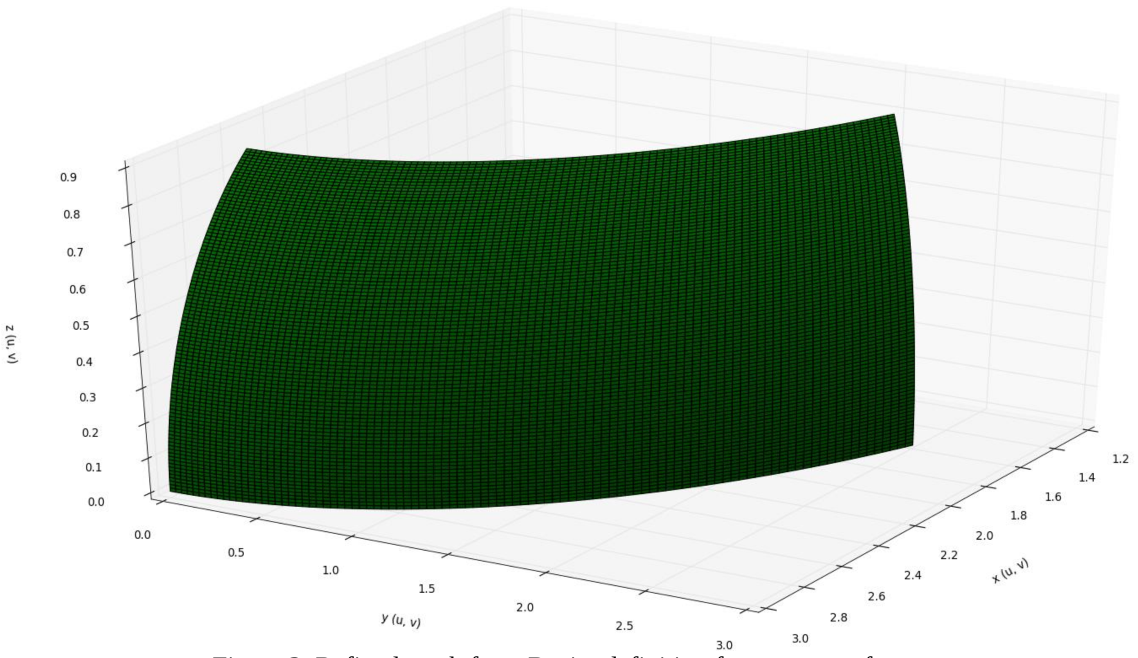

Figure 3. Refined mesh from Bezier definition for a torus surface segment.

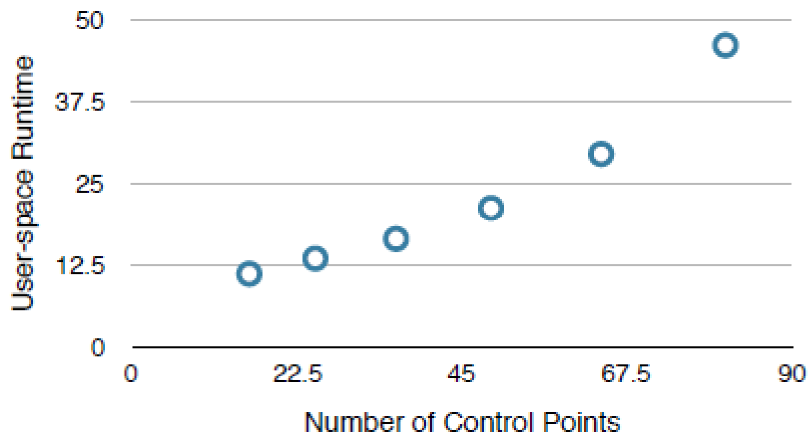

Figure 4. Run time vs. number of control points $(n_u+1)(n_v+1)$, corresponding to a range of [3:8] for the Bezier surface order.

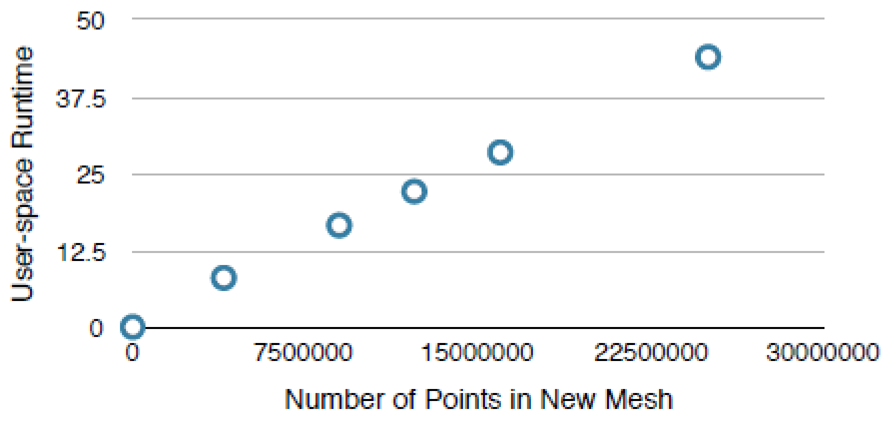

Figure 5. Runtime vs. Grid mesh size.

B. **Sphere-Cone Geometry**

After surface testing, the method was applied to volumes and rewritten into a series of modules in the AFRL Ablation Fluid Structure Interaction (AFSI) library. A grid refinement program was written where a coarse mesh grid can be read in and then refined to a level desired by user input. For this work we show the process on the front section of the sphere cone geometry. This is a single block that wraps around the spherical nose and continues down the cone of the geometry. Figure 6 shows the initial coarse grid that was the test case for the fitting algorithm. A single 38x500x150 grid block was extracted and saved to a Plot3D grid file. A solid rendering of the grid block is shown in Figure 7. The file was read in by the program and fitted to a Bezier volume of order 5 in all directions. A refined block 75x999x150 then reproduced with the Bezier volume definition. Figure 8 shows the refined grid (denoted by red lines) laying over top the original grid (denoted by black lines). It can be seen that the fitting algorithm visually reporoduces the correct interpolation for the refined grid and preserved the surface curvature similar to the torus example in the previous section of this work. The AFSI library already has shock fitting algorithms in it as part of the grid refinement process using the Bezier definitions. We then can take this Bezier grid run a coarse simulation and both refine and shock fit a solution. Figure 9 shows the application of the shock fitting method on the nose section grid block. Note the large free-steam section of the grid is pulled in close to the bow shock. This is accomplished by moving all of the control points of the Bezier grid outside the shock closer in while leaving the surface and near wall control points constant.

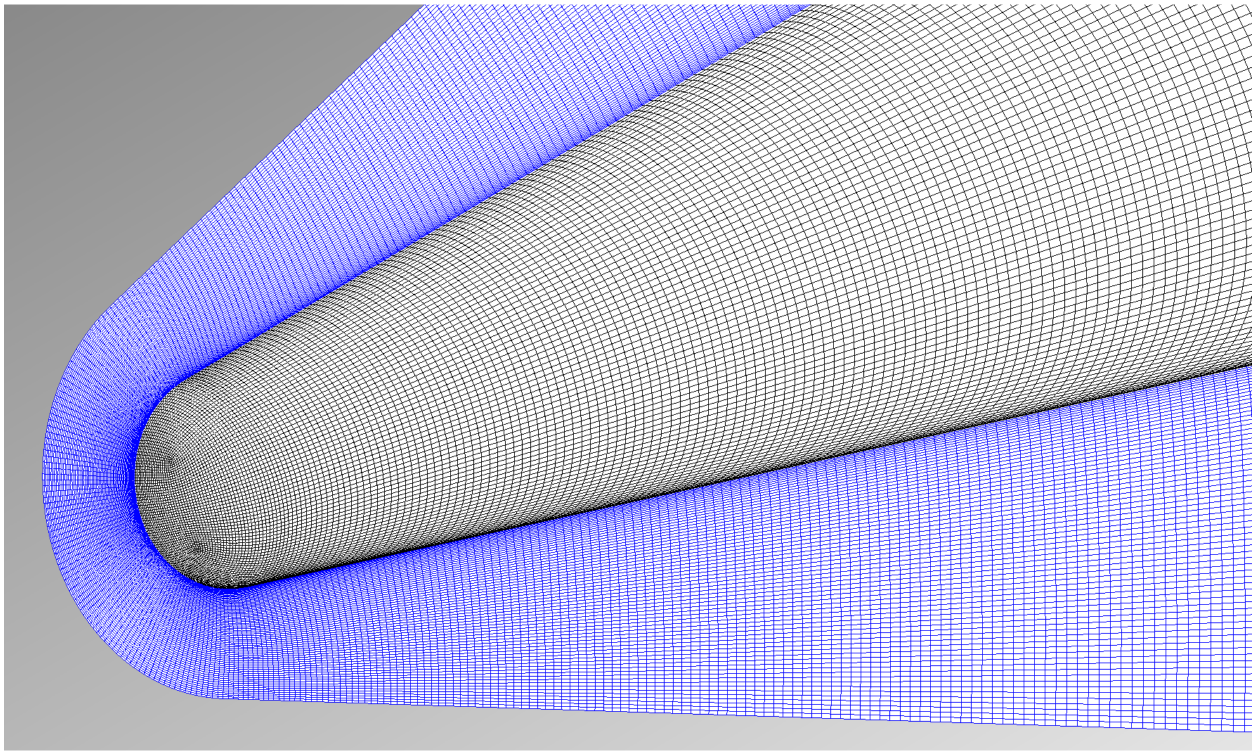

Figure 6. Coarse grid example of sphere cone geometry. Black lines are sphere cone surface mesh. Blue lines are symmetry plane surface mesh.

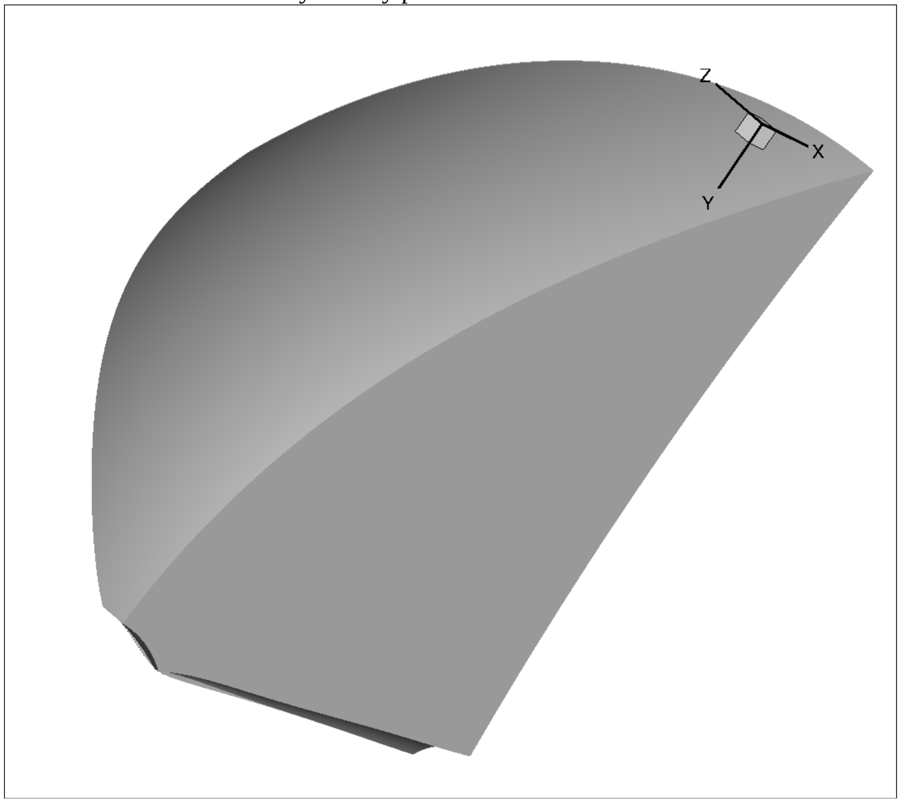

Figure 7. Solid rendering of single grid block section. (Free stream further from surface than in figure 6.)

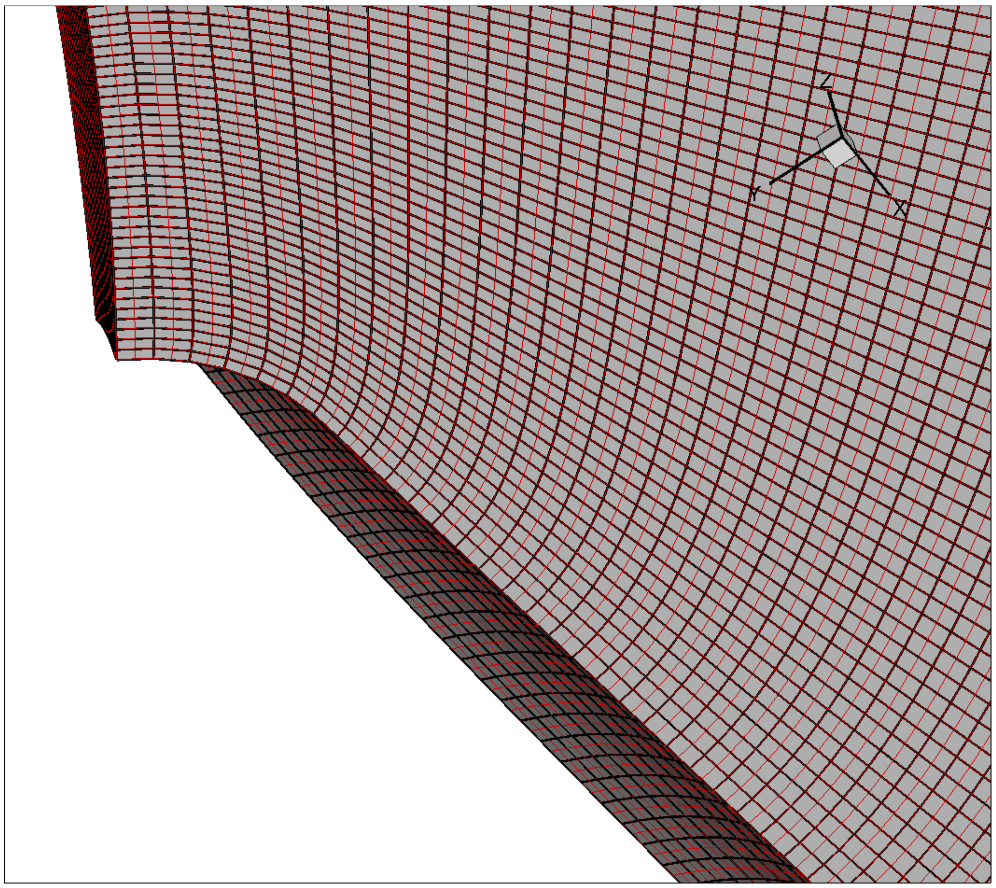

Figure 8. Coarse grid (black lines) with refined grid in tangential direction of sphere-cone geometry. Boundary layer clustering in the wall normal direction is not applied to aid in showing tangential refinement.

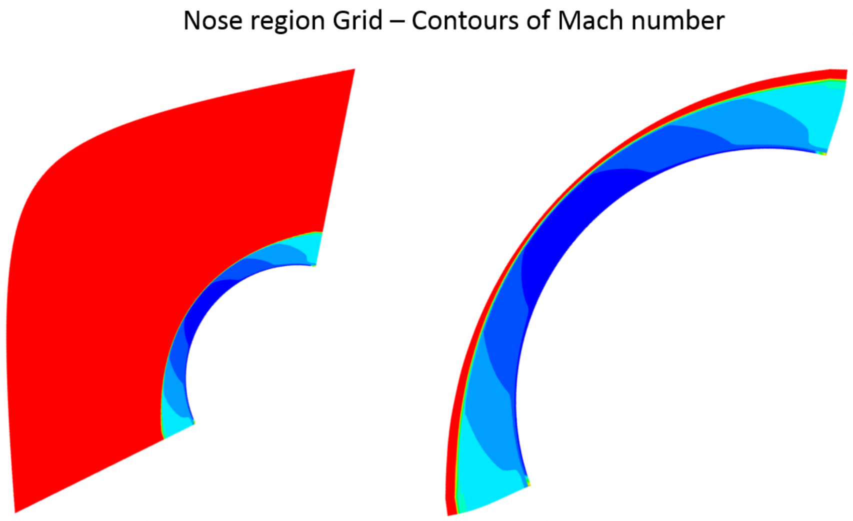

Figure 9. Planar slice of nose region of sphere-cone geometry. Initial grid (left) and final shock fitted grid (right).

C. **Conclusion**
The capability to store the definition of the grid in a compact format is an obvious advantage when it comes to data movement for large-scale calculations. The ability to locally compute the grid requirements results in much faster re-processing time than reading in a multi-TB data file describing grid systems over 100 Billion elements in size, such as the ones expected to be used when exascale HPC systems become available in the near-term future. The current work shows that the simple representation of the grid with the Bezier definition is acceptable with respect to geometry preservation and refinement control for a system of multi-block grids. Now that the ability to import grids is available in AFSI, we will focus on run-time performace of the algorithms and start to incorporate the grid metric computing algorithms. We will perform a comparison to computing second order grid transformation metrics and look at the scaling of the algorithms.

Acknowledgements
This material is based upon work supported by, or in part by, the Department of Defence (DoD) High Performance Computing Modernization Program (HPCMP), under the User Productivity, Technology Transfer and Training (PETTT) Program, contract number GS04T09DBC0017. We would like to thank the DoD Supercomputing Resource Centers for making their computational resources available to us.

References
[1]Farin, G. E., “Curves and Surfaces for Computer Aided Design: a Practical Guide”, 4th ed., Boston, Mass. U.a: Acad., 1997.
[2]Pastava, T., “Bezier Curve Fittng”, Master’s Thesis, 1998.