Ninth MSU-UAB Conference on Differential Equations and Computational Simulations.
Electronic Journal of Differential Equations, Conference 20 (2013), pp. 27–37.
ISSN: 1072-6691. URL: http://ejde.math.txstate.edu or http://ejde.math.unt.edu
ftp ejde.math.txstate.edu

FAST EVALUATION OF COMPLEX EQUATIONS OF STATE

ERIC M. COLLINS, EDWARD A. LUKE

Abstract. One of the most common operations encountered in computational fluid dynamics (CFD) solvers is the evaluation of the caloric and thermal equations of state (EoS) which are required to compute thermodynamic state variables from the conserved values that are typically being advanced by the simulation. The complexity of these calculations can vary widely depending on the nature of the fluid under consideration. We present a method for generating an interpolated representation of the EoS that is fairly inexpensive to evaluate regardless of the complexity of the actual underlying state equations. This approach has the advantage of being agnostic towards the original representation; whether it be a complex analytic expression, expensive iterative method, or interpolated from empirical data.

1. Introduction

Computational efficiency has always been one of the most important issues facing developers of high-performance computing applications. However, as high-performance computing hardware continues to improve – delivering ever-increasing FLOPs – engineers continue to crave increasingly accurate simulations based on increasingly complex physical models. Obviously, this comes with additional computational cost, but what often goes unacknowledged is that different types of model complexity contribute to computational cost in significantly different ways.

In this paper, we briefly look at some of the competing demands of efficiency versus complexity, particularly when that complexity is embedded deep within solver iteration hierarchy. We then propose a solution which could shift much of the expense required to evaluate certain aspects of the model to a pre-processing step. This step generates an approximation of the original model to some prescribed level of accuracy. A fairly simple algorithm is then employed evaluate the interpolated function within the context of the solver hierarchy.

The present technique is largely derived from several previous efforts which have utilized tabular look-up techniques to avoid costly computations [4][5]. Our proposed solution extends these techniques by offering adaptive table sizes and resolutions to satisfy user specified accuracy constraints as well as support for enforcement of the continuity of the tabular approximation function and its derivatives.

2000 *Mathematics Subject Classification.* 65D15.
Key words and phrases. Tabular approximation; adaptive sampling; Bezier surface; spline; interpolation.

Published October 31, 2013.

1.1. **Motivation.** As high-performance computing hardware continues to improve, engineers continue to push the fidelity of computational simulations to accurately model increasingly complex physical phenomena. The increased complexity of the underlying physical models can have a drastic impact on the computational solvers that implement them.

At a wide range of pressures and temperatures most gases behave according to the linear ideal gas law. The ideal gas law provides a simple closed form that relates pressure, temperature, and mixture to fluid enthalpy, density, and entropy. When subject to sufficiently high pressures and/or low temperatures, however, many gases will behave very differently. This change in behavior is often due to close range effects, such as Van der Waals forces. Under these conditions, the behavior of the gas can depart substantially from the linear model of the ideal gas, and closed form solutions may no longer exist that accurately describe the relationship between various thermodynamic variables.

For fluid conditions in which simple closed form equations are not available, non-linear iterative solvers are often required to obtain values for enthalpy, density, or entropy at a given pressure and temperature. Luke and Cinnella [2] describe solving fluid equations using the Hirschfelder, Buehler, McGee and Sutton (HBMS) EoS. These relations involve three regions with nested non-linear functions. Another method which is often employed is the NIST REFPROP EoS that uses a modified Benedict-Webb-Rubin (MBWR) EoS [6]. The MBWR EoS is described by 32 coefficients with an additional 20 coefficients used to describe saturation curve data and critical point properties.

In the context of computational fluid dynamics solvers, in which each cell or mesh point requires a solution to these complex systems of equations, the use of non-linear solvers to obtain the required thermodynamic variables can easily become the dominant computational cost. In practice, many solvers will resort to utilizing simpler, less-accurate equations simply to make the simulation computationally tractable.

In our approach, we have utilized an interpolated representation that is fairly inexpensive to evaluate, regardless of the complexity of the actual underlying state equations. This approach has the advantage of being agnostic towards the original representation, whether it be based on complex analytical expressions, expensive iterative methods, or interpolations of empirical data. The expensive evaluation operations are performed as a pre-processing step during which the underlying tabular approximation is generated. Once converted to our representation, all equations have nearly the same look-up and evaluation cost.

While our initial inspiration was drawn from the adaptive Cartesian reconstructions of Xia, Li, and Merkle [5], the underlying topological configuration of our table representation, is probably more similar to the polynomial splines over hierarchical T-meshes (PHT-Splines) of Deng et al. [1]. However, rather than using B-spline based representations, we have chosen to utilize independent Bezier patches which have been carefully reconstructed to ensure C^1-continuity at the patch edges. Our algorithm allows for anisotropic refinement of the mesh depending on the features of the underlying surface data, while allowing, at most, only one hanging node per mesh element edge.

In the following sections we describe a generic function approximation algorithm for real valued functions of two independent variables, and a method for efficiently evaluating the approximated surface. For the sake of generality, the independent variables will be referred to as x and y, while the function to be approximated will be denoted as $F(x, y)$. The actual choice of independent variables will depend on the details of a particular application. Our immediate application for this approximation method is to determine an unknown thermodynamic state variable from two known variables which uniquely define the state. For example, density or energy can be computed from pressure and temperature. We may make occasional reference to this application in the following sections as it has been the primary motivation for some of our design choices.

2. Methodology

At the lowest level, the underlying representation is a piecewise C^1-continuous cubic Bezier surface. The cubic Bezier surface is a commonly used modeling element which is found in computer graphics (CG) and computer-aided geometry design (CAGD) applications. The mathematical description of the surface is both flexible and easy to evaluate [3].

$$F(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_j^3(v)b_{ij} \tag{2.1}$$

Here, $B_i^3(u)$ are the third-degree (fourth-order) Bernstein polynomials,

$$B_i^3(u) = \frac{3!}{i!(3-i)!}(1-u)^{3-i}u^i \tag{2.2}$$

and b_{ij} are the control points. The shape of the surface is completely determined by a linear combination of these control points. The blending coefficients can be determined by evaluating the Bernstein polynomials (Eq. 2.2) at the parametric location $(u, v) \in [0, 1] \times [0, 1]$, corresponding to the desired point on the surface, or by the evaluation of an equivalent algorithm such as the deCasteljau algorithm [3].

Each cubic Bezier surface in the representation is referred to as a patch. Each patch has boundaries aligned with constant coordinate directions (e.g. constant x and y). The surface is generated by performing a recursive subdivision of the desired domain until the resulting Bezier patches are capable of interpolating the original function to a specified error tolerance or a prescribed maximum recursion depth. Depending on the nature of the function to be approximated, sampling and subdivision can be accomplished in the space of the original independent variables (x and y), or in the logarithmic space ($\log(x)$ and $\log(y)$).

Once the patches are sufficiently refined to resolve the function, we establish C^1-continuity conditions at the interface between the patches. This process involves several steps to ensure that the resulting surface is both well defined, and well behaved. Details are provided in the following sections.

2.1. **Lagrange approximation.**

During the first stage of approximation, all Bezier patches are generated by sampling the underlying function at 16 uniformly spaced locations (4×4) within the respective sub-domain of the patch. The control points for the patch are then computed such that the cubic Bezier surface will pass through the sampled points. We refer to this as the Lagrange approximation.

The control points may be found by solving a 16×16 linear system formed from the evaluation of the Bezier surface at the parametric locations of the sampled

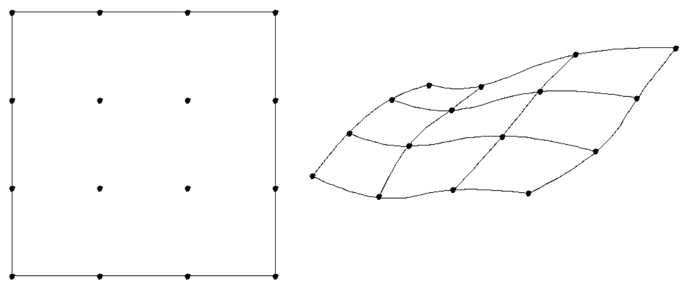
FIGURE 1. Sampling sites for Lagrange interpolation

points. Let $B_{ij}(u,v) = B_i^3(u)B_j^3(v)$, and our system is formed as:

$$\mathbf{f} = \mathbf{B}\mathbf{b}$$

where $\mathbf{f}$ is a vector of the 16 sampled values. The matrix $\mathbf{B}$ stores computed values of $B_{ij}(u,v)$. The parametric values for (u,v) are fixed in each row according to the location at which the corresponding sample point $f \in \mathbf{f}$ was evaluated. And finally, $\mathbf{b}$ is a vector of 16 control points which uniquely define the surface. If we utilize a 2D indexing scheme to reflect the 2D nature of both the sample mesh and the control net, the matrix and vectors can be expanded as follows:

$$\begin{bmatrix} f_{00} \\ \vdots \\ f_{mn} \\ \vdots \\ f_{33} \end{bmatrix} = \begin{bmatrix} B_{00}(u_0,v_0) & \cdots & B_{ij}(u_0,v_0) & \cdots & B_{33}(u_0,v_0) \\ \vdots & \vdots & \ddots & \vdots & \\ B_{00}(u_m,v_n) & \cdots & B_{ij}(u_m,v_n) & \cdots & B_{33}(u_m,v_n) \\ \vdots & \vdots & \ddots & \vdots & \\ B_{00}(u_3,v_3) & \cdots & B_{ij}(u_3,v_3) & \cdots & B_{33}(u_3,v_3) \end{bmatrix} \begin{bmatrix} b_{00} \\ \vdots \\ b_{ij} \\ \vdots \\ b_{33} \end{bmatrix}$$

Here, the f_{mn} values correspond to the 16 sampled values which are located at the parametric coordinates (u_m, v_n):

$$u_m = \frac{m}{3}, \quad v_n = \frac{n}{3}$$

and the mapping from parametric to state space is given by:

$$x = (1-u) * x_{lo} + u * x_{hi}, \ y = (1-v) * y_{lo} + v * y_{hi}$$

with bounds of each patch from (x_{lo}, y_{lo}) to (x_{hi}, y_{hi}).

To obtain the control points, we invert the system matrix $\mathbf{B}$ and apply it to both sides.

$$\mathbf{B}^{-1}\mathbf{f} = \mathbf{b}$$

Since the Bernstein polynomials are independent of both the sampled values and the control points, the entries in the system matrix $\mathbf{B}$ are constant. Therefore, $\mathbf{B}$ and $\mathbf{B}^{-1}$ should only need to be evaluated once.

2.2. **Recursive refinement.**

The recursive subdivision step is carried out by choosing a split-plane in the x or y direction. The splits are computed so as to divide the patch's subdomain in half. In the x-direction, the split is computed using:

$$\bar{x} = \frac{1}{2}(x_{hi} + x_{lo}).$$

The split in the y direction is computed in a similar manner. For each candidate split, two pairs of Bezier patches are generated - one on either side of the split plane. The pair which reduces the error in the approximation the most is the one selected for the refinement. This approach allows for anisotropic refinement of the domain in the direction where it is needed most first.

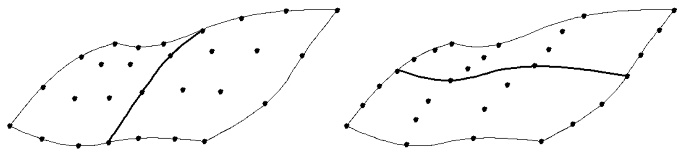

FIGURE 2. Candidate patch splits with recomputed sample sites

Notice that the recomputed sample points along shared edges are evaluated from the original function definition. When these sampled values are used as input into the Lagrange approximation algorithm, the resulting surface definitions produce a common shared edge approximation.

This patch splitting process continues until each patch satisfies a prescribed local error tolerance. Alternative stopping criteria can also be provided. For example, a maximum number of recursive refinements can be specified to prevent the algorithm from excessive refinement near features such as singularities or discontinuities. The recursion halts once a patch either satisfies the convergence criteria, or it exceeds the stopping criteria.

The error in the patch approximations are evaluated by performing a numerical integration using tenth-order Gauss-Legendre quadrature. Technically, only a fifth or sixth order quadrature should be required to properly account for the leading error terms. However, we felt that the extra sampling resolution was justified to ensure that no important features were left unresolved during this refinement process. More importantly, care must be taken to ensure that the locations chosen to sample the error for the numerical integration do not coincide with the locations where the function was sampled to generate the approximation. It is in these sampled locations that the Bezier surface approximation will exactly interpolate to the sampled function, which would result in the integrated error norms being inappropriately biased.

2.3. **Balance criteria.**

After the initial refinement process has completed, the resulting set of patches form a piecewise discontinuous surface as a collection of fully independent cubic Bezier surfaces (i.e. each patch has their own unique set of control points). As previously mentioned, any patches which share a common edge with exactly one other patch will have interpolated to the same four sampled values on that edge, and will therefore be C^0-continuous along that edge. However, any patch that shares an edge with more than one adjacent patch is not guaranteed to possess this property. It is likely that the approximated surface will be discontinuous along those edges.

Before we can establish C^1 continuity throughout the surface, we must first attempt to minimize the effects of sampling bias by reducing the amount by which adjacent patches are allowed to differ in size. A balancing criteria has been established whereby all patches are required to be within one level of refinement of each of its neighbors. One consequence of this criteria is that all hanging nodes will only appear on the mid-points of patch edges. In other words, each patch will have at most two adjacent neighboring patches along each of its edges. Thus the sampling rate for adjacent patches will vary by a factor of two at most.

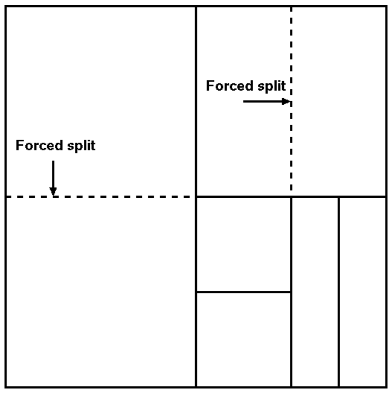

FIGURE 3. Patch splits are required to satisfy the balance criteria.

The set of patches which resulted from the initial recursive refinement steps are now further refined until all patches satisfy the balance criteria. Each of these subdivided patches are once again fit with cubic Bezier approximations to the original function using the Lagrange interpolation method described above. Since the balance step consists only of patches being subdivided, the accuracy of the approximation will most likely be improved. Thus, the ability of the patches to satisfy the error tolerance criteria is not effected. In addition, the most refined patches will, by definition, not have any adjacent neighbors which are more refined. Therefore, this process will not create any patches below the lowest refinement level, preserving the recursion limit if one has been specified. In other words, neither of our halting criteria is violated by this step.

2.4. **Hermite approximation.**

In addition to the Lagrange approximation technique, we also make use of a second technique for generating cubic Bezier surface patches. We refer to this second method as the Hermite approximation. Rather than attempting to fit the patch to 16 uniformly sampled values, the function and its derivatives at the corner nodes are utilized. More specifically, this method uses the value of the function (F), its first derivatives with respect to each of the independent variables (F_x, F_y), and the cross-derivative (F_{xy}), evaluated at each corner node.

In our implementation, the values at the corner nodes are computed for each patch from the Bezier patches that were generated from the Lagrange approximations during the initial refinement steps. That is, rather than use derivatives derived from the original function (which may or may not be readily available), we utilize the derivative information encoded within the existing Bezier interpolations.

2.5. **Reconciliation.**

Since the derivative data is computed independently for each patch, there may be a multiplicity of values at shared corner nodes. This node information is then reconciled amongst the adjacent patches in a two-step process.

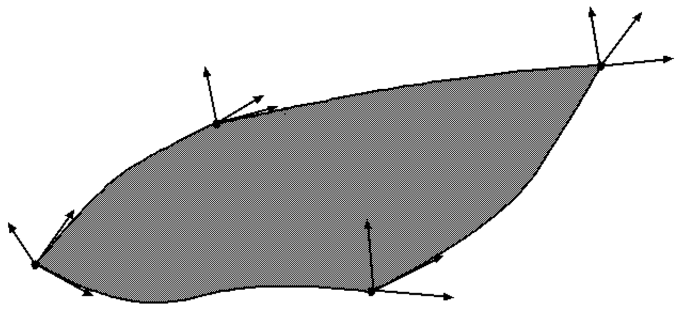

FIGURE 4. Hermite approximation from the function value and its derivatives provided at the corner nodes

Step 1: For any node which is solely a corner node (i.e. is not a hanging node for any adjacent patch), we account for the following observations: (a) The function values should all already agree since the corner nodes are among the sixteen original sampled points in each patch. Therefore, they should all already be evaluating to the exact value of the original function.

(b) To minimize the potential for the approximation to oscillate near rapidly changing features (e.g. discontinuities or singularities), we prefer to use the derivative with the smallest magnitude. This typically has the net effect of smoothing the approximated surface. (See figure 5)

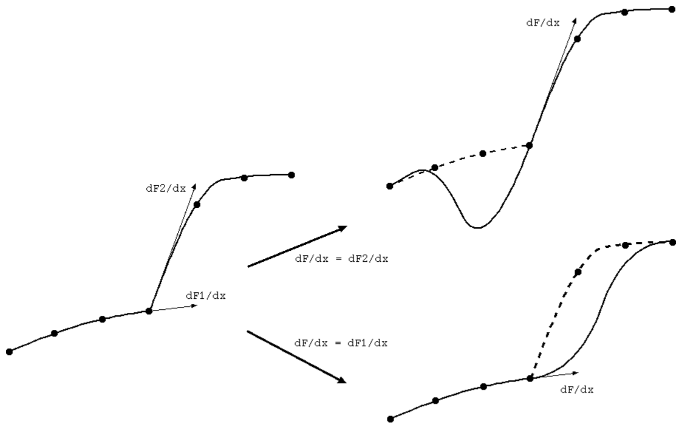

FIGURE 5. Slope limiting to reduce oscillations

Step 2: For the remaining nodes, we note that each hanging node is a hanging node for exactly one patch, and a corner node for the two neighboring patches. Since we desire the entire edge be C^1-continuous, we must insist that the two neighboring patches are generated in such a way that the edge shared with their less refined neighbor matches in the C^1 sense. To accomplish this we specify that the value of the function, and its three derivative values at the hanging node must be taken from the less refined patch. In other words, we evaluate the value and the derivatives of the less refined patch on the edge midpoint corresponding to the location of the hanging node. This information is then passed on to the two adjacent patches as corner node data for their respective interpolations.

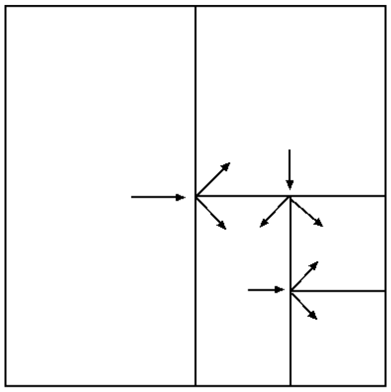

FIGURE 6. Hanging node dependency graph

Since it is likely that the shape of many of the patches will be necessarily altered by this approach, there is a specific order in which the updates to the patch shapes must be made. For this purpose, we form a data structure which records the data dependencies between all patches and hanging nodes. The data is then topologically sorted to obtain the proper ordering for the update operations.

After all of the patches have been regenerated using the technique described above, the collective surface approximation should be C^1-continuous at all points within the specified domain.

2.6. **Evaluation.** Once we have established the piecewise C^1-continuous Bezier patch approximation to a given function, all that remains is to develop a fast method for evaluating an arbitrary point on this surface. There are two steps to the evaluation process: to search for the patch whose domain includes the point of interest, and then to evaluate the patch at that point. To these ends, we employ a *kd*-tree search algorithm, and the well-known deCasteljau evaluation algorithm.

For the *kd*-tree search, we take advantage of the fact that the patches are organized hierarchically. Every time we split the domain during the refinement phases above, we also created a natural partitioning of the patches. This implies that for any node at any level of the tree, the sub-domain spanned by patches beneath that node is a contiguous rectangular region completely covered by the patches found in the child nodes beneath the current node.

The *kd*-tree is generated by sorting the patches into a nested array based on the current split-direction and value. There are four possibilities regarding the distribution of child patches. If there is only one child cell, then it is considered a leaf node. If there are two or more child nodes, then there exists at least one split through the center of the sub-domain for which all child patches exist solely on one side or the other. The patches can therefore be partitioned by an x-coordinate split, a y-coordinate split, or both. If both directions are possible (i.e. no single patch spans the width or height of the remaining sub-domain), then the patches are sorted by splitting in the direction perpendicular to the orientation of the parent node in the tree. In other words, wherever possible, the orientation of the split will alternate from one level of the tree to the next.

The look-up algorithm thus proceeds by simply comparing the coordinates of the desired point with the current split value and direction until a leaf node is reached. At that point, the local coordinates within the patch are computed, and then the Bezier patch is evaluated. The mapping from global to local coordinates is given by:

$$u = \frac{x - x_L}{x_R - x_L} \tag{2.3}$$

$$v = \frac{y - y_L}{y_R - y_L} \tag{2.4}$$

where (x_L, y_L) are the lower bounds of the patch, and (x_R, y_R) are the upper bounds.

The deCasteljau algorithm proceeds by recursively evaluating linear combinations of adjacent control points within the control net. In 2D, this evaluation has the form:

```
for $k$ from 2 to 0
  for all $(i,j) \in [0,k] \times [0,k]$
    $b^k_{i,j} = (1-u)(1-v)b^{k+1}_{i,j} + u(1-v)b^{k+1}_{i+1,j} + (1-u)vb^{k+1}_{i,j+1} + uvb^{k+1}_{i+1,j+1}$
```

where $b^3_{i,j}$ are the sixteen original control points, and $b^0_{0,0}$ is the evaluation of the desired point on the surface.

3. Results

In high pressure environments – such as those occurring in rocket engines, deep underground, or in certain industrial processes – complex non-linear equations of state (EoS) are often required for the solver to make realistic predictions. Evaluation of these relations often requires an expensive iterative solver. The formulation of some EoS may also utilize expensive operations such as exponential and power functions. As a result, most of the computation time required to obtain solutions to these problems is spent evaluating the EoS.

Using the method described above, we have been able to replace these expensive EoS evaluations with evaluations of our adaptive tabular surface approximations. We are able to obtain independent approximations the density and internal energy functions down to a specified error tolerance. Derived quantities, such as specific heats (c_p, c_v) and sound speed, can also be computed from these values and their derivatives obtained from the tabular fits. Since the evaluation of the cubic Bezier patch is generally much cheaper than the evaluation of complex equations of state, these surface approximations can dramatically improve the speed of computations with only modest loss in accuracy.

To demonstrate the effectiveness of the proposed approach we compare the performance of our tabular equation of state against the NIST standard properties EoS known as REFPROP [6] described earlier. Surface approximation tables are constructed for the super-critical region (with temperatures greater than the critical temperature) over the valid evaluation region for the REFPROP EoS of three different materials: CO_2, O_2, H_2O. The tables were refined to within about 0.1% relative accuracy or better. Our method was able to achieve this goal with around $2,000$ patches for each of the species. An illustration of the tabular fit for CO_2

density is shown in figure 7. The highly enriched region near the bottom right corner of this table is the region near the critical point of the fluid where density changes rapidly.

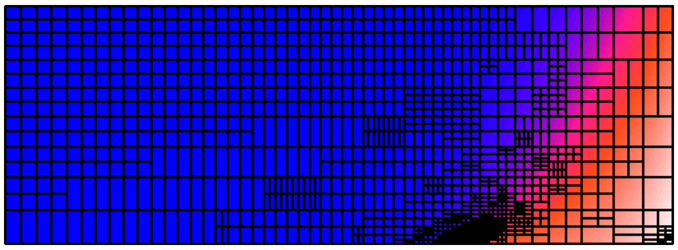

FIGURE 7. Adapted table structure for density of super-critical CO_2.

To compare the performance of our tabular EoS implementation to the REFPROP EoS, timing data was obtained for the evaluation of 200,000 points sampled along the diagonals of the fit space. The results of this performance comparison are shown in table 1. For these tests, our adapted tabular EoS implementation achieves a performance speedup over the REFPROP EoS by approximately two orders of magnitude.

TABLE 1. Comparison between NIST REFPROP and Tabular EoS

Model	Table Size	Error	REFPROP Time	Tabular Time	Speedup
CO_2	1968	0.011%	16.23s	0.0786s	206
O_2	2082	0.046%	7.10s	0.0793s	89
H_2O	1949	0.110%	14.32s	0.0783s	183

It is important to note that the errors of the NIST REFPROP EoS are generally on the order of 0.1% when compared to experimentally measured data. This highly efficient tabular implementation can achieve a significant performance increase with hardly any significant losses of accuracy.

We also note that the isotropic tabular formulation of Xia et al. [5] required 225,121 patches to fit the super-critical region of CO_2 from the REFPROP database to an accuracy of only 0.1%. Our table was able to achieve a lower error bound with just 1,968 patches. The improved efficiency of the tabular representation can be attributed to the anisotropic refinement afforded by our tabular fitting technique. Xia et al. reported performance improvements over REFPROP for their method that were similar to our observed speedups.

Conclusion. In this report, we have outlined an approach to generate interpolated look-up tables for functions of two independent variables. This technique may be deployed whenever the evaluation of one or more sufficiently complex functions begin to consume a significant fraction of the computational resources for a given simulation.

By selecting a simple representation for the underlying surface approximation, the evaluation of complex functions and their first derivatives can be accelerated.

In practice, we have observed that our technique can deliver a factor of two to three speed up in the context of multi-species chemically reacting flow simulations involving detonations and blast wave physics.

As an additional benefit, this technique now provides a common interface for the flow solver to evaluate complex equations of state, regardless of their original source. We are now able to run simulations with EoS that were previously prohibitively expensive to evaluate or were in a form that would be difficult to incorporate into our solver in an effective manner.

The extension of this technique to functions involving more independent variables is, at first glance, fairly straight-forward. However, the complexity of the refinement, the search algorithms for finding the desired patch, and the evaluation of the resulting n-dimensional hyper-surface may be rather more computationally challenging to implement efficiently. We are investigating application areas where such an approach may be warranted.

References

[1] Jiansong Deng, Falai Chen, Xin Li, Changqi Hu, Weihua Tong, Zhouwang Yang, and Yuyu Feng. Polynomial splines over hierarchical T-meshes. *Graphical Models*, 70:76–86, 2008.

[2] Ed Luke and Pasqualle Cinnella. Numerical simulations of mixtures of fluids using upwind algorithms. *Computers and Fluids*, 36(10):1547–1566, 2007.

[3] Les Piegl and Wayne Tiller. *The NURBS book*. Monographs in visual communications. Springer, 1997.

[4] F. Douglas Swesty. Thermodynamically consistent interpolation for equation of state tables. *Journal of Computational Physics*, 127(1):118–127, August 1996.

[5] Guoping Xia, Ding Li, and Charles L. Merkle. Consistent properties reconstruction on adaptive Cartesian meshes for complex fluids computations. *Journal of Computational Physics*, 225:1175–1197, 2007.

[6] Ben Younglove and Mark McLinden. An international standard equation of state for the thermodynamics properties of refrigerant 123 (2,2-dichloro-1,1,1-trifluoroethane. *Journal of Physical Chemistry Reference Data*, 23(5):731–779, 1994.

Eric M. Collins
Center for Advanced Vehicular Systems, Mississippi State University, Box 9627, Mississippi State, MS 39762, USA
E-mail address: `emc@cavs.msstate.edu`

Edward A. Luke
Computer Science and Engineering, Mississippi State University, Box 9637, Mississippi State, MS 39762, USA
E-mail address: `luke@cse.msstate.edu`