Multivariate Functional Approximation of Scientific Data

Tom Peterka, Youssef Nashed, Iulian Grindeanu, Vijay Mahadevan, Raine Yeh, and David Lenz

1 Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA
2 Mathematics and Computer Science Division, Argonne National Laboratory
(Current address Stanford Linear Accelerator Laboratory, Stanford University, CA, USA)
3 Purdue University, West Lafayette, IN, USA

1 Abstract

While our computational capability to generate raw data grows, the ability to store, transform, and draw conclusions from scientific data is lagging. Beginning with the introductory chapter and continuing throughout much of this book, we see numerous examples of how in situ processing can help close the gap between data generation and data analysis. This chapter expands the discussion of in situ methods beyond when and where data are processed, to how data are represented. Rethinking the way that scientific data are represented can empower subsequent visualization and analysis, especially when such data transformations are performed in situ.

Scientific data may be transformed by recasting to a data model fundamentally different from the discrete pointwise or element-wise datasets produced by computational models. In Multivariate Functional Approximation, or MFA, scientific datasets are redefined in a hypervolume of piecewise-continuous basis functions. Compared with existing discrete models, the continuous functional model can save space while affording many of the same spatiotemporal analyses without reverting back to the discrete form.

In this chapter, modeling the MFA, in situ, is presented. The data model and modeling approach are parallelized for high-performance computing. A lightweight and efficient method of enforcing high-degree continuity across subdomains in the parallel decomposition is also included. The MFA can subsequently be used post hoc to evaluate points and derivatives anywhere in the domain, facilitating numerous analysis and visualization applications.
2 Introduction

Three observations motivate the approach described in this chapter. The first two reasons are common to other in situ methods in this book. The growing disparity between computational and I/O rates with each new generation of supercomputer requires more data analysis and visualization to be conducted at the source of the data. For example, the rate of data that can be computed on the Summit supercomputer at the Oak Ridge Leadership Computing Facility (assuming 1 byte generated per clock cycle) is five orders of magnitude greater than the bandwidth of its parallel file system. Beyond performance, in situ analyses can also improve the quality of science. While the fidelity of post hoc analysis is hamstrung by the low temporal frequency of writing data to persistent storage, in situ analysis can have much higher fidelity because analysis tasks have access to all the data directly, not only the tiny fraction saved for postprocessing.

The third motivation, specifically in this chapter, addresses the type of processing to conduct in situ. In addition to reducing data size, the objective is to transform the data model into a more useful form for analysis. Both in situ and post hoc, many operations involving managing, analyzing, and visualizing scientific data can be streamlined when data are in a continuous functional form, rather than a discrete set of points. Converting discrete data into piecewise-continuous basis functions multiplied by a sparse set of control points results in a hypervolume of NURBS or B-splines. This transformation is called Multivariate Functional Approximation, or MFA. The objective of this chapter is to represent scientific data in situ, in an MFA model, and to use that model instead of the original discrete dataset for post hoc analysis and visualization.

The MFA model can represent numerous types of data because it is agnostic to the mesh, field, or discretization of the input dataset. Compared with existing discrete data models, the MFA model can enable many spatiotemporal analyses, without converting the entire dataset back to the original discrete form. The MFA often occupies less storage space than the original discrete data do, providing some data reduction, depending on data complexity and intended usage. For example, noise may be intentionally smoothed by using a small number of control points and high-degree basis functions; alternatively, high-frequency data features may be preserved with more control points and lower degree. Post hoc, the MFA enables analytical closed-form evaluation of points and derivatives, to high order, anywhere inside the domain, without being limited to the locations of the input data points.

A key decision in functional data analysis is the choice of basis function family. Fourier, wavelet, and geometric (i.e., spline) bases are commonly used; recently Austin et al. proposed the Tucker decomposition as a low-rank alternative. Majdisova and Skala proposed radial basis functions for particle data. The MFA uses geometric (specifically, nonuniform rational B-spline) bases because they mirror the space-time properties in the original data and retain...
these geometric properties in analytics and visualization. Computing geometric bases requires minimal memory overhead, and the parallel decomposition of the original space-time domain is identical in the geometric functional domain, an important consideration for minimizing data movement at scale.

NURBS (nonuniform rational B-splines) are piecewise-continuous, are differentiable, have local support, and are invariant to affine transformations. In order to compute the MFA in $d$ dimensions, a $d$-dimensional tensor product of NURBS bases is computed. Both field geometry (space-time vertex positions) and science variables (pressure, density, temperature, etc.) are modeled. The model is efficiently represented by a set of control points and knots. The control points are reference points that “push and pull” the representation through their linear combination with basis functions. The knots map partitions of the data to the control points and associated basis functions.

The MFA is designed to model scientific data in situ, in parallel, on high-performance computing (HPC) platforms. The model is adaptively refined until a user-set error threshold is achieved, and it includes a lightweight and efficient method of enforcing high-degree continuity across subdomains in the parallel decomposition, to the same degree of continuity as the blocks’ interior MFA models (Figure 1.1).

What makes the MFA unique is that the transformed data retain geometric properties: spatial and temporal contiguity, derivatives, and statistical distributions are all preserved. The goal of analyzing scientific data—which are inherently spatiotemporal—is usually to understand the relationship of science variables to their position in space and time. Thus, once data are modeled in the MFA, many data analyses and visualizations are possible directly from it.
Two fundamental operations are required in an MFA: building a model from an input discrete dataset and evaluating points and derivatives from the model. The former is usually performed in situ; the latter is often post hoc. Both stages are covered in this chapter. Following a discussion of related work in the next section, in situ modeling is presented in Section 4 and post hoc uses of the MFA are explained in Section 5. Parallel implementations of both modeling and evaluation are covered in Section 6. The chapter concludes with a look at some scientific use cases and a glimpse at the future of functional approximation for scientific data.

3 Related Work

Multivariate functional approximation borrows ideas from several fields: compression, statistics, modeling, visualization, and analysis. Piecewise functional approximations replace discrete data points with linear combinations of basis functions and a small number of reference points called control points. Statisticians call this method functional data analysis with low-dimensional serial implementations available in popular statistics packages.

NURBS are used extensively within computer-aided design (CAD) software tools. Fitting NURBS to existing data is not straightforward, however, because the inclusion of rational weights results in a nonlinear problem. Hence, most NURBS implementations resort to using uniform weights (all set to 1.0), which are then manually tweaked by users for additional model shape control. This is the approach taken in the MFA as well. Rational weights are implemented and supported, but for performance reasons the weights are currently set to 1.0 (i.e., reverting to B-splines).

Previous research demonstrated that B-splines can faithfully represent scientific data in up to three dimensions: 2-d triangular surfaces and 3-d tetrahedral meshes can be converted into bivariate and trivariate models. Martin and Cohen first developed the data model and framework in 2001, and Martin et al. described how to parameterize triangular and tetrahedral data as tensor products in 2008 for modeling exterior surfaces and interior volumes of medical datasets.

Geometric functional representations can replace data models used today in visualization algorithms. Martin and Cohen derived isosurfacing, slicing, and ray tracing algorithms directly from B-spline models. Raviv and Elber showed direct rendering from trivariate B-splines for isosurface extraction, planar slicing, and volume rendering. Park and Lee visualized trivariate NURBS for flow data. Hua et al. modeled 3-d solids using simplex splines and showed that visualization algorithms such as isosurfacing and volume rendering can operate on the same data model.
Functional data models are used in mechanical and fluid engineering simulations. One example is isogeometric analysis (IGA) [24], which uses NURBS models for mechanical simulations. Recently, a parallel IGA toolkit [11] built on PETSc [4] was developed for solving high-order partial differential equations over NURBS basis domains. Spectral methods are another example of a discretization that uses basis functions to evaluate data: the Nek5000 Navier-Stokes solver [14] is based on a weighted sum of basis functions defined over a coarse set of control points.

Such engineering simulations usually take as input a mesh that was created in a CAD system, and NURBS modeling is a rich and mature topic in CAD literature. Lin et al. [28] summarized numerous approaches to reverse engineering NURBS surfaces given a set of 3-d input points. Partition of unity methods [3] allow local approximation to extend to the entire domain by blending basis functions of local approximations into a global solution with the desired continuity across boundaries of local regions.

The local regions of a global model can be organized in various ways. Sederberg et al. [49] introduced T-splines to conserve space and to ensure continuity along B-spline patches. The idea is to localize the addition of knots and control points in refined regions, rather than extending the refinement in all dimensions over the entire domain, as in a traditional tensor product representation. T-splines, however, increase the bookkeeping effort to keep track of the irregular T-shaped connectivity of refined regions.

\section*{4 In Situ Modeling of the MFA}

We begin by introducing the minimum background, consisting of common notation and a few equations, needed to understand the rest of the chapter. Afterwards, we describe the method to build an MFA model approximating an input dataset to a desired tolerance; first with a fixed number of control points and then with an adaptive scheme where the resolution of the model varies with complexity of the dataset.

\subsection*{4.1 Mathematical Background}

A curve (Figure 1.2) can be parameterized as a vector-valued function of a single parameter $u$ such that

$$C(u) = \sum_{i=0}^{n-1} N_{i,p}(u)P_i. \quad (1.1)$$
Fig. 1.2 Left: a B-spline curve. Right: a tensor product of B-spline curves in a surface. Tensor products extend to higher-dimensional hypervolumes.

$N_{i,p}$ are the $p$th degree basis functions, and $P$ is the set of $n$ control points; $n$, the number of output control points, is usually less than $m$, the number of input data points. The definition extends to higher manifold dimensions (surfaces, volumes, hypervolumes) as a tensor product of multiple parameters $u_1, u_2, ..., u_d$. The right panel of Figure 1.2 shows a surface, and in general a $d$-dimensional hypervolume is parameterized as follows:

$$V(u_1, ..., u_d) = \sum_{i_1} \cdots \sum_{i_d} N_{i_1,p}(u_1) \times \cdots \times N_{i_d,p}(u_d) P_{i_1,...,i_d}. \quad (1.2)$$

The basis functions are computed by using the recurrence formula of Cox [10] and De Boor [12]. The recursive computation of $N_{i,p}(u)$ requires computing $O(p^2)$ nonzero coefficients. The degree $p$ is a small number, usually between 1 and 10.

The $n$ control points $P$ are found by solving a linear least-squares optimization problem $(N^T N)P = R$, where $R$ is computed from the $m$ input data points and basis functions in $O(m + n)$ time [27]. The matrix of basis functions $N^T N$ (in normal form) is $n^2$ in size, positive definite, and sparse with $2p + 1$ nonzero entries along the diagonal [13]. The vector of control points can be solved without pivoting in $O(n^3)$ time.

4.2 Modeling with Fixed Size and Separable Dimensions

In this section, we describe the basic steps to model the MFA: specifically, how to model the dimensions separately in a multidimensional domain, thereby reducing the computational complexity and memory footprint. In the separable-dimension approach, the memory to compute the matrix of basis functions is allocated only once for each new dimension and is reused; hence, memory size does not grow with the number of dimensions.
Understanding the steps needed to build an MFA model that approximates a set of $m$ input points is easiest if one begins with a fixed number of control points, $n$. In addition to $n$, the user specifies the polynomial degree, $p$, for the model. The first step is to compute parameter values in the range $[0.0, 1.0]$ for all the input data points. In the current MFA implementation, parameters are assigned to input points according to the spacing of input points in each of the principal domain dimensions, specifically by translating and scaling each input point by the extents of the domain in each dimension. Other parameterizations based on chord length between points, and other metrics, are possible. In general, however, the parameterization problem is ill-posed for arbitrary input points. Piegl and Tiller [44] provided a comprehensive overview of parameterization approaches.

Knots are the breakpoints in the parameter space between different basis functions. The number of knots per dimension is $n + p + 1$ by definition. Absent any local adaptive refinement (the subject of the next section), the default knot placement for the internal knots is to follow the distribution of the parameters of the input points. Both the parameters and the knots are represented by one vector per dimension, with the total number being the sum over the dimensions rather than the product.

Instead of fitting all the dimensions simultaneously, it is less expensive to decompose the dimensions and fit each dimension separately. In the work of Peterka et al. [41], the approach of Piegl and Tiller [44] was extended to any number of dimensions, as illustrated in Figure 1.3 for 3-d input points. Assume that the number of input points is $(m_x, m_y, m_z)$ in the $x, y, z$ dimensions and that the number of control points is $(n_x, n_y, n_z)$. The original data are approximated as a set of 1-d curves in the first dimension, for example, the $x$ direction. The number of curves is the product of the number of points in the remaining dimensions, $m_y \times m_z$. The linear least-squares system described in the preceding section is solved for each curve in $O(n_x^3)$ time.

Before solving in the second dimension, the $n_x$ control points for each solved curve in the first dimension replace the input data points. The input points now consist of fewer points (assuming fewer control points than input points), specifically $n_x \times m_y \times m_z$. We now take curves in the $y$ dimension, and to each curve we fit a set of control points by solving the same linear system, of $n_y$ control points in $O(n_y^3)$ time, for each curve. There are $n_x \times m_z$ number of those curves.
**Fig. 1.4** Overall adaptive refinement algorithm.

Once again, the resulting control points replace the input data points, becoming the new input for the third dimension. The new input data points now consist of $n_x \times n_y \times m_z$ points. Taking curves in the $z$ direction and fitting $n_z$ control points for each of the $n_x \times n_y$ curves, the resulting control points are the final solution.

The method extends to any number of dimensions, and the result is the same as if a full-dimensional system were solved in one step. However, the cost of performing separate steps, solving for 1-d curves in each dimension, is lower. Assuming that the number of domain dimensions is $d$ and that the number of control points is $n$ in each dimension, solving the full-dimensional least-squares problem in a single step would take $O(n^{3d})$ time complexity and $O(n^{2d})$ space complexity. By comparison, the separable method takes $O(n^d + n^2)$ time complexity and $O(n^2)$ space complexity, producing identical results in shorter time and less space. The savings of the separable method over the full-dimensional method improve as the dimensionality grows.

### 4.3 Local Adaptive Refinement

A model built on a fixed number of control points and predetermined knot spacing, without taking features in the data into account, cannot represent complex input signals accurately or efficiently. An alternative is to begin with some initial set of control points and knots, as described above, and then adaptively add more detail as needed. The user provides the allowable error, $e_{\text{max}}$, for example, the maximum relative error normalized by the extent of the data values. Other error metrics such
as the L2-norm or root mean squared (RMS) error are possible with no loss of generality.

Figure 1.5 illustrates the steps to adaptively fit the MFA, refining regions of high error until the error everywhere is below a user-prescribed threshold $e_{\text{max}}$. Beginning with an initial knot distribution for the minimum number of control points, control points and knots are adaptively added until all the evaluated points in each span of knots are within $e_{\text{max}}$ of the original points. Knot spans with error greater than $e_{\text{max}}$ are subdivided, and the MFA is recomputed.

Recomputing all the control points anew during every iteration is inefficient because a global solution and evaluation of the error at every input point are required during every adaptive iteration. Because the global solution procedure ignores the results of the previous iteration, the global procedure solves the entire domain of input points from scratch each time. To address these shortcomings, Nashed et al. [37] developed a local adaptive algorithm that is able to incrementally refine an existing model without recomputing the MFA over the entire input domain or compromising the continuity of the model.

This local incremental method takes advantage of B-spline local support to refine regions of the approximated model, acting locally on both input and model subdomains, without affecting other regions of the global approximation. The method is diagrammed in Figure 1.5 for $p = 3$ and proceeds as follows. Insert a knot where the error is highest, by removing $p - 1$ control points (red circles, left panel of Figure 1.5) and adding $p$ new control points (black circles, center panel), without changing the decoded curve/surface. Perform a local fitting (right panel of Figure 1.5) for the new control points (black circles) with equality constraints for $p$ boundary control points (red circles). Repeat for the next highest error location.

This approach reduces the computational burden by restricting the iterative optimization locally in subdomains of both the approximation and the input domains, and it naturally lends the algorithm to a parallel implementation. In contrast, the global refinement method performs global modeling and evaluation operations in
Fig. 1.6 Comparison of local refinement with global refinement in terms of resulting compression factor on the left and solution time on the right.

each iteration, which eventually become more expensive than those in the local refinement method. Figure 1.6 shows that the local refinement algorithm results in higher compression (left panel) and lower time to solution (right panel) once the input size is large enough, compared with the global algorithm.

The local adaptive refinement method is directly generalizable to high dimensions and is able to model scientific data, as shown in Figure 1.7. The adaptive spacing of control points that results is clearly visible, as well as the relationship between the control point spacing in the upper portion of the figure and the varying complexity of the data in the lower portion.

The adaptive method and the resulting accuracy of the model are driven by a user-supplied error bound. At present, we apply this error bound only to values, not to derivatives, when fitting the model. However, one could modify the adaptive algorithm described above to control the error of derivatives or some combination of values, derivatives, and other application-specific constraints, if desired.

4.4 Modeling Scientific Data

An assortment of scientific use cases demonstrates how the MFA can be used to fit actual scientific datasets consisting of turbulence, shocks, and high-frequency variation. Exemplar applications include combustion, computational fluid dynamics, turbulent mixing, and global climate. Data relating the size of the MFA and error of the approximation, along with an image of reconstructed points evaluated from the MFA model, are presented for each dataset. The relative RMS error reported is the root mean squared error normalized by the extent of the data range.
**Turbulent combustion.** The turbulent combustion dataset is generated by an S3D simulation [8] of fuel jet combustion in the presence of an external crossflow [20][18]. The domain is 3-d \((x, y, z)\) \((704 \times 540 \times 550)\), and the range variable \(f(x, y, z)\) is the magnitude of the 3-d velocity. Figure 1.8 shows a volume rendering of the resulting evaluated points from an MFA modeled with \(p = 3\).

**Thermal hydraulics.** A 3-d vector field representing the numerical results of a large-eddy simulation of Navier-Stokes equations for the MAX experiment [35] is representative of turbulent mixing and thermal striping that occur in the upper plenum of liquid sodium fast reactors. The data, generated by the Nek5000 solver, have been resampled from their original topology onto a \(200 \times 200 \times 200\) regular grid, and the magnitude of the velocity vector is associated with each 3-d domain point. Figure 1.9 shows a volume rendering of reconstructed data when the MFA is modeled with \(p = 3\).

<table>
<thead>
<tr>
<th>Input Data Points</th>
<th>Output Control Points</th>
<th>Relative RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2.1 \times 10^8)</td>
<td>(1.4 \times 10^7)</td>
<td>(4.6 \times 10^{-4})</td>
</tr>
</tbody>
</table>

Fig. 1.7 A 2-d slice of S3D turbulent combustion data modeled by the MFA. Below, the surface evaluated from the model. Above, the mesh of control points.

Fig. 1.8 S3D dataset showing velocity magnitude evaluated from the MFA model.
Rayleigh-Taylor instability. Rayleigh-Taylor instability \([30]\) occurs at the interface between a heavy fluid overlying a light fluid, under a constant acceleration, and is of fundamental importance in a multitude of applications ranging from astrophysics to ocean and atmosphere dynamics. Small perturbations at the interface between the two fluids interact nonlinearly and eventually become turbulent. The dataset was generated by the CFDNS [29] Navier-Stokes solver. One time step of \(288 \times 512 \times 512\) velocity vector magnitude data was modeled with \(p = 3\), and Figure 1.10 shows a volume rendering of the reconstructed data from the MFA.

Atmospheric climate. The Community Earth System Model (CESM) is a global climate model providing data of the Earth’s oceans, atmosphere, land, and sea ice. The dataset in this example is the FLDSC (clear-sky downwelling longwave flux at
surface) variable of the Community Atmosphere Model developed at the National Center for Atmospheric Research [38], and the data lie on an $1800 \times 3600$ 2-d domain with one value of FLDSC at each grid point. Figure 1.11 shows the evaluated points when modeled with $p = 3$.

The MFA, in contrast to floating-point lossy compression methods, is a transformed data model that retains geometric properties and enables analytical operations directly. Rather than reducing the number of bits used to encode the input data points, the adaptive MFA fitting algorithm automatically selects the number and location of control points. The resulting accuracy and size of the model depend on the complexity of the data, as the previous examples show. The MFA and lossy compression can be seen as complementary methods with different objectives. The two methods can be combined; for example, compression can further reduce the storage size of MFA control points and knots. In other instances, one or the other method may be more appropriate depending on the purpose of the in situ operation and its post hoc uses.

5 Post Hoc Use of the MFA

The MFA is a surrogate data model that is solved once (typically in situ) and then used many times for data analysis and visualization post hoc, facilitating applications beyond simply reproducing the input data points. The MFA is designed to be used directly for subsequent data analysis and visualization, distinguishing it from other basis representations such as wavelets, cosine transformations, and compression algorithms that require the inverse transform to be applied first. Several operations are possible directly from the MFA without reverting to the original discrete data model. Moreover, they are possible in the full order and accuracy of the model, without linear interpolation or finite difference estimation. While the model itself is an approximation to the original data, subsequent applications of the MFA are
exact, analytical, high order, and closed form. No additional error is introduced when evaluating or differentiating points from a model that has previously been approximated.

Once the model is computed in situ, one can evaluate the model at any point in the domain, not just at the original input points, and differentiate the model up to \( p - 1 \) times in any combination of partial derivatives in the domain dimensions. Because the model is intrinsically high-order, all point evaluations are done at the same high order or degree to which the model is fitted. Affine transformations can be applied directly to the control points. Statistical and machine learning algorithms using the MFA, such as clustering, topological analysis, and principal components analysis, are areas of further research. The following sections explain point evaluation and differentiation—basic building blocks for post hoc usage of the MFA—followed by applications of these building blocks.

5.1 Multidimensional Point Evaluation

Points in any dimensionality are evaluated by computing Equation [1.2] Figure [1.12] shows an example of evaluating a 2-d point with \( p = 2 \) in both dimensions. The number of basis functions and control points multiplied in each dimension is \( p + 1 \); hence, 9 control points are shown in image (a). First, each curve in the first (horizontal) dimension is collapsed into a single point by the matrix multiplication of basis functions and control points. This step is shown in images (b) and (c). Next, the three resulting points form a curve in the vertical direction, which is collapsed in (d) through multiplication of basis functions to become the resulting evaluated point in (e).

This algorithm, which extends naturally to any number of dimensions, has two main steps: computing the basis functions and multiplying basis functions by control points. The basis functions are computed in \( O(p^2) \) time, and the control points are multiplied by basis functions in \( O((p + 1)^d) \) time, in \( d \) dimensions.
5.2 High-Order Differentiation

To compute derivatives from a previously modeled MFA, one needs only to differentiate the basis functions, replacing \( N_{i,p} \) with \( N'_{i,p} \) in Equation 1.2. Because the basis functions are \( p \)-degree polynomials, they are differentiable \( p \) times except at knot locations, where they are differentiable \( p - 1 \) times. The derivatives of a \( p \)-degree basis function are computed with a similar recurrence formula as the basis function values, in \( O(p^2) \) time. In fact, the original basis functions for the values can be thought of as the zero-degree derivatives. For example, the basis functions for \( p = 3 \) polynomials and their first two derivatives are shown in Figure 1.13.

Derivatives of the MFA use the same knots and control points as the original MFA does, allowing the MFA to be computed once in situ and stored in a file. Basis functions and their derivatives are not stored; instead, they are recomputed as needed. A single model, consisting of knots and control points, allows all the values and all derivatives (total, partial, and mixed) to be available post hoc. The only requirement is to model the original MFA with high enough degree \( p \) that any derivatives needed later are less than or equal to \( p - 1 \) in degree.

5.3 Applications

High-order point evaluation and differentiation enable a multitude of visualization and analysis operations. Some applications are possible directly from the control points and knots of the MFA, while others take advantage of being able to evaluate and differentiate points (sample the data) at arbitrary positions, in closed form. The MFA can be used to visualize high-order data, to compute Jacobian and Hessian fields, as a surrogate representation in machine learning algorithms, to compare and remap data points from one discretization to another, and to smooth data and filter noise.
High-order visualization. High-order data such as those resulting from finite element and spectral methods have traditionally been problematic for scientific visualization algorithms such as isosurfacing, slicing, and volume rendering. Solutions were limited to either linearly approximating high-order elements—still the most common approach—or calling custom functions provided by the simulation. Linear approximations introduce error, while custom callbacks require complex software interconnections between simulations and visualization, which are difficult post hoc when the simulation no longer is running. The MFA solves this problem because high-order evaluation of data points in any resolution is a closed-form operation with no additional error. In fact, all of the visualizations in this chapter were generated by evaluating a dense set of points from high-order MFA models of at least degree three and then applying existing visualization tools. In the future, the algorithms in those tools, such as volume rendering, can be written to ingest the MFA model directly, essentially moving the MFA point evaluation to be inside the visualization algorithm.

Derivative fields. Differentiation, including high-order derivatives, is a staple of visualization and data analysis. Gradient fields, velocities, Jacobian matrices, edge detection, topological segmentation, and uncertainty quantification all require first derivatives. Second derivatives are used for computing curvature, acceleration, and Hessian matrices. For example, ridge and valley features are defined in terms of gradients and eigenvalues of the Hessian $\mathbf{H}$. Applying lighting and shading to ridge features requires their normal, or the third derivative of the original model. Smooth derivatives are also required for optimization algorithms, used in machine learning (segmentation, classification, and regression, e.g.) that search for minima along directions of steepest descent. Figure 1.14 demonstrates computing a gradient field, showing evaluated points in the upper image and showing in the lower image the gradient field where the $z$ coordinate is the first partial derivative of the MFA with respect to $x$. Both the upper and lower images were computed post hoc from
the same previously computed and saved MFA model. It is easy to visually confirm
that the slope in the top image corresponds to the value in the lower image. Other
combinations of derivative degree (second derivative, e.g.) and partial and total
derivatives are possible to compute from the same model post hoc.

**Comparison of datasets with different discretizations.** The ability to evaluate data
points at arbitrary locations, not limited to locations of the input dataset, facilitates
several analysis operations. One such operation is to compare two datasets whose
resolutions or, more generally, discretizations differ. Discretization here includes
element type, order, and resolution. Each of the input datasets can be modeled in a
separate MFA, and then the two MFA models can be evaluated at the same parameter
locations. Because the MFA is meshless, the field data of two heterogeneous datasets
can be compared at uniform locations, independent of the mesh representation of
each dataset.

**Remapping between coupled multiphysics simulations.** A related problem making
use of high-order arbitrary evaluation is the transferring of a field from one
discretization to another, as occurs in coupled multiphysics simulations. This process
is usually called remapping. A further challenge in remapping is to satisfy physical
constraints such as conservation of integrals and derivatives, for example, total
energy of a system or flux across an interface. The linear least-squares method of
solving the MFA can be replaced with nonlinear constrained optimization—as was
done in Section 4.3—with penalty terms added for the physics constraints. Once
modeled, points are evaluated from the MFA as described above.

**Smoothing, simplification, filtering.** Smoothing, simplification, and filtering are
related operations that extract the underlying signal from a noisy dataset. Data
arising from sensors and experiments usually contain noise that can mask the
underlying behavior, which is often more smooth and has lower frequency than
the raw data have. The MFA provides smoothing capability, being built from
piecewise-continuous functions of high order. Compared with mesh simplification
and topological simplification methods, the MFA is agnostic to mesh discretization
and topology. Simplification or smoothing is accomplished by decreasing the number
of control points while optionally increasing the degree of the MFA. These are
common geometric editing operations of NURBS and B-splines [43]. Figure 1.15
demonstrates smoothing of a synthetic dataset [32] with different degrees of input
noise, by using a high degree of $p = 10$ and half as many control points in
each dimension as input points. The MFA in the lower row retains more of the
low-frequency structure of the data than the input points in the upper row, where
the structure is masked as the amount of noise increases. Alternatively, instead of
being smoothed, high-frequency features could be retained by reducing the degree
and increasing the number of control points (not shown).
6 Parallel Approximation and Evaluation

Computation of the MFA is parallelized on three levels. First, block parallelism (i.e., data parallelism) is used to decompose the domain into blocks and execute each block in distributed-memory compute nodes of a supercomputer or computing cluster. The DIY library [36] provides block parallelism. Within a block, task parallelism is utilized to approximate curves and evaluate points by using thread-level tasks. Currently, TBB [42] is the threading model used for CPU threading, but other task-parallel programming models can be used. For example, a SYCL [26] kernel is under development to parallelize the evaluation of multiple point locations, which can run on a GPU as well as on a CPU. Linear algebra operations, for example, to invert matrices, are SIMD (single instruction, multiple data) vectorized by using the Eigen [21] linear algebra library.

Each block of the parallel data model is a tensor product defined by $n$ control points and $n + p + 1$ knots in each dimension. Control points are in the same coordinate system as original data points are, and therefore the same spatiotemporal domain decomposition used for input data can be reused for the MFA, minimizing data movement. The basis functions are not stored and are recomputed as needed.

The output MFA is stored in a binary file in DIY format, which is read and written in parallel by using MPI-I/O. There is also a serial utility to convert the file to VTK format so that the results are compatible with visualization and analysis tools derived from VTK such as ParaView [2] and VisIt [9]. In future work, it is anticipated that those tools will read the DIY file in parallel, and VTK filters will be written to operate directly on the MFA. One of the reasons for choosing a NURBS geometric basis for the MFA is its compatibility with such downstream tasks.

When computing an MFA approximation in parallel over multiple blocks in a spatial domain decomposition, the interior of each block will be $C^p$-continuous, $p$ being the polynomial degree, but discontinuities exist across neighboring block boundaries.
Fig. 1.16 Left: blocks modeled independently are discontinuous at the block boundaries. Right: blending using high-degree smoothstep functions restores continuity of the desired degree across block boundaries.

Grindeanu et al. [19] developed an efficient and scalable solution to this problem that involves blending neighboring approximations to ensure $C^p$ continuity across block boundaries. They showed that after decomposing the domain in structured, overlapping blocks and approximating blocks independently to the desired accuracy, the local solution can be extended post hoc to the global domain by using compact, multidimensional smoothstep functions. This approach, which can be viewed as an extended partition of unity approximation method, is scalable on HPC architectures.

The global domain is first decomposed into rectangular blocks, which are extended by a fraction (ghost zone) to overlap the neighboring blocks. The MFA is then computed locally, in situ, independently for each extended block. The only difference between this step and the modeling algorithms presented earlier is that the blocks of input data are slightly enlarged by the ghost zone; otherwise, the modeling is identical.

Post hoc, additional communication is involved when evaluating points from the MFA. Blocks send requests for evaluated points in ghost zones to neighboring blocks and receive the evaluated points from their neighbors. The evaluated points from multiple blocks that intersect in a ghost zone are blended together by using a smoothstep function, selected from a family of functions of varying degrees $p$, depending on the desired continuity $C^p$. These functions [16], used in computer graphics and visualization, are simple to evaluate, since they involve only polynomials of degree $2p + 1$. The value of the smoothstep function, $\alpha$, is used in 1-d to blend two points $P_1$ and $P_2$ as $\alpha P_1 + (1 - \alpha) P_2$. By induction, Grindeanu et al. [19] extended the blending to higher dimensions and also proved that the error, with respect to the input data, of the blended point in the ghost zone is guaranteed to be within the same bounds as the user-specified error that was used to model the block interiors.

Blending occurs independently per dimension, with different blending functions and different ghost zone sizes possible in each dimension. The number of points being blended together depends on the dimensionality of the domain and on the number of neighboring blocks meeting at a face or corner. The result is a multidimensional partition of unity of the corresponding point evaluations that satisfies the same
degree of continuity and same error bounds as the block interiors. Figure 1.16 shows a close-up of the junction between 4 independent blocks, before and after blending. Point evaluations are local operations per block, and blending is also a local operation after the values are received from neighboring blocks. For structured data, the communication pattern is predictable and localized to exchanging information only between neighboring blocks.

In order to evaluate the cost of the blending, parallel scaling studies were conducted on the Bebop cluster at Argonne’s Laboratory Computing Resource Center. Bebop has 1,024 compute nodes, with Intel Broadwell processors having 36 cores and 128 GB of memory per node, connected by an Omni-Path fabric. Figure 1.17 shows a strong-scaling study on an S3D scientific dataset. The turbulent combustion dataset generated by an S3D simulation of fuel jet combustion in the presence of an external crossflow is 3-d (704 × 540 × 550); the field variable is the magnitude of the 3-d velocity, and $p = 3$ was used to model the MFA. The experiment measures how much time the post hoc point evaluation takes with and without blending, in parallel, using a previously computed and saved MFA model. Results show that blending added a factor of approximately 2X to the cost of evaluating, and the strong-scaling efficiency is 54% at 2,048 MPI processes.

7 Ongoing and Future Work

The techniques presented in this chapter are freely available for others to test and use in their own research, and community contributions are welcome.

1 Freely available at https://github.com/tpeterka/mfa
Research efforts continue along several fronts. Currently, the MFA for each block is computed in situ independently, without continuity constraints, and it is only during the evaluation of points from the MFA post hoc that continuity is achieved by blending evaluated points from neighboring blocks. An alternative approach is being investigated that enforces continuity constraints while neighboring MFA blocks are being solved in situ.

Research also is ongoing to determine an optimal set of knots—both number and location—based on properties of the discrete data. The choice of knot vector influences the resulting accuracy of the approximation. Methods to automatically determine a knot vector that achieves high approximation quality are being developed. At the core of the approach is a feature function that characterizes the amount and spatial distribution of geometric details in the input data by accumulating derivatives. Knots are then selected to evenly distribute the feature contents across their intervals. A solution to this problem in 1-d has recently been published \[50\], with future extensions to higher-dimensional and unstructured input data being pursued.

Also being developed is an alternative data model representation for the MFA based on T-splines. This is an adaptive model, analogous to adaptive mesh refinement, consisting of regions with various levels of adaptivity (numbers of knots and control points). In the current tensor product representation, every time a new control point is added, it is duplicated in every dimension of the hypervolume, essentially adding a hyperplane of control points in each dimension. The T-spline formulation sidesteps this requirement, at the expense of added complexity of the data model and potentially increased cost of computing the MFA and evaluating points from it. It is anticipated that the storage savings will be a favorable trade-off given the added complexity.

Acknowledgments

This work is supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, program manager Laura Biven. This research used resources of the Argonne Leadership Computing Facility (ALCF), which is a DOE Office of Science User Facility supported under Contract No. DE-AC02-06CH11357, the Argonne Laboratory Computing Resource Center (LCRC), and the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.
References


23