Eurographics Symposium on Parallel Graphics and Visualization (2018)
H. Childs, F. Cucchietti (Editors)

Interactive Visual Analysis of Multi-dimensional Metamodels

S. Gebhardt, S. Pick, B. Hentschel, and T. W. Kuhlen

Visual Computing Institute, RWTH Aachen University, Germany

Abstract

In the simulation of manufacturing processes, complex models are used to examine process properties. To save computation time, so-called metamodels serve as surrogates for the original models. Metamodels are inherently difficult to interpret, because they resemble multi-dimensional functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that map configuration parameters to production criteria. We propose a multi-view visualization application called memoSlice *that composes several visualization techniques, specially adapted to the analysis of metamodels. With our application, we enable users to improve their understanding of a metamodel, but also to easily optimize processes. We put special attention on providing a high level of interactivity by realizing specialized parallelization techniques to provide timely feedback on user interactions. In this paper we outline these parallelization techniques and demonstrate their effectivity by means of micro and high level measurements.*

Categories and Subject Descriptors (according to ACM CCS): Simulation and Modeling [I.6.6]: Simulation Output Analysis—Numerical Analysis [G.1.1]: Interpolation—Interpolation Formulas Computer-Aided Engineering [J.6]: Computer-aided manufacturing (CAM)—

1. Introduction

The understanding and configuration of manufacturing processes is a complex task, influenced by a multitude of input parameters. E.g., quality criteria, such as the roughness of a cut for a laser cutting process, are affected by configuration parameters, such as the focus of the laser beam. These parameters are usually chosen from technology tables that provide information on the results of specific pre-measured parameter settings. This, limits the choice of parameter combinations to exactly these pre-measurements.

Computational process simulations can be used to overcome this limitation. However, each simulation run may take hours or even days. To reduce computation time, global approximation models are constructed as surrogates for the original models. These surrogates are referred to as *metamodels*, because they are an abstraction of the original model. They represent the mapping of a process' parameter space to its criteria space as function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Due to their multi-dimensional nature, they are inherently hard to understand.

In this paper, we propose *memoSlice*, a visualization application designed to facilitate the interactive visual exploration of multi-dimensional metamodels of manufacturing processes. Specifically, *memoSlice* has been developed in close collaboration with domain scientists who investigate laser cutting processes. Our development was guided by the following two challenges. First, we needed to devise a visualization that facilitates the accessible exploration of the underlying function space. Second, production-size metamodels can be quite large; their interactive exploration requires an efficient evaluation scheme in order to ensure fast user feedback.

We address the first challenge by following a coordinated multiple views (CMV) approach, which tightly integrates several visualization techniques (see Fig. 1 / Sec. 4). In order to meet the second challenge, we propose a highly dynamic task-parallel execution scheme that focuses on immediate user feedback rather than shortest overall execution time. This scheme is detailed in Sec. 5. We evaluate our approach's performance in a set of micro benchmarks and end-to-end measurements, which we discuss in Sec. 6. Finally, we conclude with a discussion and an outlook on future work.

2. Related Work

The analysis of multi-dimensional data is a core area of visualization. However, no solution has been presented so far that suits all needs for analysis of multi-dimensional metamodels for manufacturing processes. Kehrer et al. survey visualization techniques for both, multi-dimensional and multi-variate data [KH13]. HyperSlice [vWvL93] is adapted as one element of our proposed visualization approach, but enriched with additional information. Interactive navigation in the parameter space was previously realized, based on the basic idea of the scatterplot matrix [EDF08]

With regard to the terminology of a recent conceptual framework for *visual parameter analysis* [SHB*14], our solution provides analysis tasks for *optimization*, *uncertainty*, and *sensitivity*

DOI: 10.2312/pgv.20181098

on the predicted outputs of *surrogate models* for the physical models of manufacturing processes. Several solutions incorporate *mutliple linked view designs* [Rob07] for *visual parameter analysis* and address similar analysis tasks as we do. HyperMoVal [PBK10] provides a combined visualization of a regression model and validation data. A solution for finding suitable parameter combinations for image segmentation algorithms is introduced [TWSM*11]: Gaussian process models are iteratively refined and explored to identify parameter combinations. An approach for the uncertainty-aware exploration of continuous parameter spaces is proposed [BPFG11]: 2D scatterplots and parallel coordinates are employed, augmented with additional information on the uncertainty of predicted results between sampling points. Vismon [BMPM12] is a design study, tailored to the needs of analysis in fishery management: the effects of two input variables on many output variables are analyzed, thereby incorporating sensitivity and constraint-based analysis as well as analysis of tradeoffs and uncertainty.

The *gradient field* is of key importance for the understanding of a scalar function [Max70, Sma61]. Specifically, for more than one input variable, *gradient trajectories* facilitate a better understanding of the behavior of a scalar field and its local sensitivity. Therefor, we include a corresponding visualization in our approach. In scientific visualization, topological approaches have enjoyed significant success in various settings, including, e.g., fast and robust iso-contour extraction and volume rendering [PSBM07, WDC*07, CSvdP10]. In particular, recent developments based on Morse theory [Mil63] have led to approaches for the computation and visualization of the Morse-Smale complex of scalar fields [EHNP03, BHEP04, GBHP08]. Ideas rooted in topology are adapted to the exploration of multi-dimensional data [OHJS10, GBPW10]. These and related approaches have been proven to be a versatile tool for the investigation of scalar functions. They are used, e.g., to track and analyze burning structures in turbulent combustion processes [BWP*10]. A generic query language defines and visualizes features based on the MSC [GKK*12]. Metamodels for production processes are often widely monotonic or of low polynomial order for individual parameters. Moreover, ideal parameter settings are not always characterized by local extrema. Hence, we rely on gradient trajectories for user guidance created by tracing integral lines [MLP*10].

3. Input Data

The data to be analyzed with our approach are metamodels for simulations of manufacturing processes [AKES15]. Their purpose is to provide a fast and cheap, yet precise approximation for the computation-intense simulation.

In general, metamodels resemble functions $f : \mathbb{R}^n \to \mathbb{R}^m$, with $\mathbb{R}^n$ being the *parameter* space and $\mathbb{R}^m$ being the *criteria* space. Parameters in $\mathbb{R}^n$ are process parameters, such as the *laser power* in a laser cutting process. In contrast, criteria in $\mathbb{R}^m$ are quantifications of process results, such as the *roughness* of the resulting cut.

The metamodel implementation relies on radial basis function networks (RBFNs) [Orr96], which provide non-linear interpolation between training points that are gathered by an iterative smart sampling approach [AKES15]. An RBFN is a three layer feed forward neural network. It comprises an input layer, a hidden layer of non-linear basis functions, and an output layer. While the input layer can be modeled as a vector of real numbers, the output layer contains a scalar function of the input layer.

The output of a metamodel exactly matches the training data when evaluating at a training point's location in the data space. Between training points, uncertainty about the quality of the result exists due to interpolation. As a measure for the uncertainty of the evaluation, the *training point density* is included as separate criterion in every metamodel.

The output of an RBFN for a given input vector $\vec{x}$ is

$$f(\vec{x}) = \sum_{i \in R} \omega_i \phi(\|\vec{x} - \vec{x}_i\|^2) \tag{1}$$

where R is the set of basis functions, ω_i is the weight of the i-th basis function, $\phi(\rho)$ is the RBF kernel and $\vec{x}_i$ is the position vector of the i-th RBF. In the implementation, three kernel types exist. These are a Gaussian kernel ϕ_g, a multi-quadratic kernel ϕ_m, and a linear kernel ϕ_l:

$$\phi_g(\rho) = e^{-\frac{\rho}{r^2}}, \quad \phi_m(\rho) = \frac{\sqrt{r^2 + \rho}}{r}, \quad \phi_l(\rho) = \sqrt{\rho} \tag{2}$$

For $\phi_g(\rho)$ and $\phi_m(\rho)$, r is the kernel width of the data field.

Not every parameter combination yields a feasible process result. Considering, e.g., a laser cutting process, any parameter combination that results in not being able to cut through the material is considered infeasible. To account for this, a second metamodel—the classification model—splits the data space in the feasible and the infeasible domain [Al 15]. Since it represents the working limits of a process, it is called *limits metamodel*.

4. Visualization Concept

As introduced above, the first challenge in order to support end users with an accessible visualization tool for metamodel exploration is to create a suitable visualization design. To this end, *memoSlice* is designed as a coordinated multiple views (CMV) system. Its consists of four main views, whose combination provides insights on different abstraction levels. As illustrated in Fig. 1, these views are the *scatterplot matrix view*, the *parallel coordinates view*, the *HyperSlice view*, and the *3D view*. They are accompanied by an optional *cut shape view*, which is present for cutting tasks.

The scatterplot matrix view and the parallel coordinates view visualize a shared set of randomly sampled data points. They mainly provide an overview of the data and allow for investigating high-level relationships, such as interdependencies between parameters and individual sensitivities of parameters on criteria. In contrast, the HyperSlice view and the 3D view facilitate detailed analysis and optimization tasks. To this end, they show slices of different dimensionalities through the multi-dimensional data domain. Additional information, i.e., *points of interest* (POI) and *gradient trajectories*, are superimposed to add further user guidance.

The visualization primitives in *memoSlice* are based on a set of shared concepts, which we describe in the following. Afterwards, we detail the individual views.

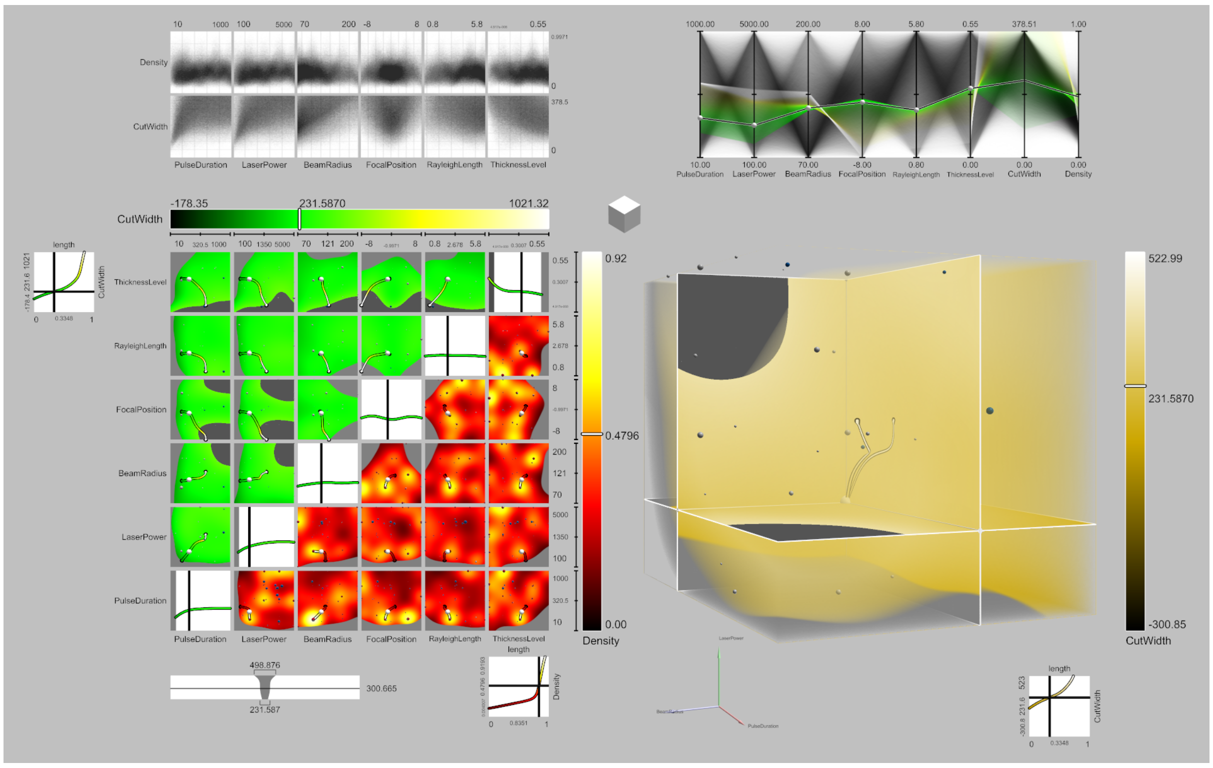

Figure 1: memoSlice*: scatterplot matrix view (top left), parallel coordinates view (top right), HyperSlice view (bottom left), cut shape view (below), and 3D view (bottom right).*

4.1. Shared Concepts

The views of a CMV application need to be coordinated, i.e., linked. In *memoSlice*, we realize this linkage via *focal points* (cf., e.g., [vWvL93, TSDS96, PBK10]). A focal point $\vec{p} \in \mathbb{R}^n$ represents a parameter combination based on which visualization primitives determine which sub spaces to display. Upon changing it, all affected visualizations are updated.

Slices are used by the visualization primitives of the HyperSlice view and the 3D view. In general, a slice is an m-dimensional cut through the n-dimensional data domain with $m < n$. In *memoSlice*, 1D, 2D, and 3D slices are used, where 1D slices are represented as function plots, 2D slices are represented as color-coded squares, and 3D slices are represented via direct volume rendering.

To aid navigation and sensitivity analysis, a projection of the *gradient trajectory* passing through the current focal point is overlayed on every 2D and 3D slice and on the parallel coordinates. For the computation of the gradient $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$, the representation of the data as RBFN enables us to rely on analytical computation of the weighted sum of RBFs $f : \mathbb{R}^n \to \mathbb{R}$ as described in Eq. 1. The gradient trajectory $\tau : \mathbb{R} \to \mathbb{R}^n$ is defined as

$$\frac{d\tau}{dt} = \nabla f, \quad \tau(t_0) = \vec{p}, \quad \phi'(\rho) = \frac{d\phi}{d\rho} \tag{3}$$

$$\nabla f(\vec{x}) = 2 \sum_{j \in N} \vec{e}_j \sum_{i \in R} \omega_i (x_n - x_{i,n}) \phi'(\|\vec{x} - \vec{x}_i\|^2) \tag{4}$$

where N are the dimension indices, with $\vec{e}_j \in \mathbb{R}^n$ being the orthonormal basis vectors, R being the set of basis functions, ω_i being the weight of the i-th basis function, $\phi(\rho)$ being the RBF kernel, and $\vec{x}_i$ being the position vector of the i-th RBF. Based on ∇f, we compute the individual vertices of a gradient trajectory by applying forward and backward integration, starting at the focal point $\vec{p}$, thereby utilizing the Runge–Kutta (RK4) method [Kut01] and adaptive step widths.

Gradient trajectories indicate the steepest ascend and descent in the data field, which corresponds to the direction where a change would have the strongest effect. Thus, they support process optimization and local sensitivity analysis.

Trajectories are color-coded with the same color map as the slices they are overlayed on to further support the understanding of their path. This way the local rate of change is depicted, which is shown even clearer in the gradient trajectory navigators above and below the HyperSlice view and below the 3D view (see Fig. 1). They show the slope of the trajectory by mapping its length to the x-axis and its value to the y-axis. Additionally, they facilitate multi-dimensional navigation, by dragging the respective sliders along the course of the trajectory, which updates the location of the used focal point.

With the scatterplot matrix view and the parallel coordinates view, we include two different point visualizations. While such visualizations can be directly used to display the training data from which metamodels are created, this is often of limited use due to a rather sparse sampling of training points. Exploiting the capability of fast metamodel evaluation, we create a *common point set* on the fly. This way, parameter combinations that lie between training points can be displayed. It also allows for applying zooming and filtering, while guaranteeing representations with consistent density.

Points that are uniformly sampled within the visible data domain allow for getting an overview on the data. Furthermore, they assist users in drawing conclusions regarding global sensitivity of parameters on criteria. To also account for local sensitivity analysis, we furhter support normal distribution sampling around the current focal point with a user-defined standard deviation (SD). This way, the direct vicinity of the focal point can be analyzed in multiple dimensions simultaneously.

Knowing the *working limits* of a manufacturing process is of key importance, since they indicate parameter configurations that do not yield feasible results. To clarify the working limits, areas with infeasible parameter combinations are grayed out in all visualizations.

Two types of *POI* are displayed as additional landmarks to improve orientation in the parameter space: locations of training points and local extrema. Both are displayed as circular overlays in the slices of the HyperSlice view and as spheres in the 3D view. Training points are displayed in light gray, local minima in blue, and local maxima in red. While training points are an inherent element of metamodels, extrema are computed in a pre-process via randomly seeding gradient trajectories in the data domain.

To allow for detailed analysis, the visible ranges for all parameters can be adjusted individually, which corresponds to zooming. Adjusting the visible range is a global configuration that affects all visualization primitives in *memoSlice*.

When considering optimization tasks, it is beneficial to users to exclude undesired criteria values from their analysis. To account for this, we include the option to limit the visible range for criteria, too. This results in points of the common point set to be discarded if their criteria values do not reside in the desired range. Similar to working limits, such regions are grayed out in slice visualizations, which reduces visual clutter and allows users to focus on desired criteria values.

To enable comparisons of different criteria, the displayed criterion can be changed for the detail views, i.e., the upper and lower triangular part of the HyperSlice view and the 3D view, respectively. Consequently, we allow users to carry out multi-criteria optimizations of up to three criteria at the same time. The training point density is included as criterion in every metamodel and can be used to identify areas that are too sparsely sampled. Additionally, it is an uncertainty indicator, so displaying it in one view increases the users' awareness for the uncertainty of analysis results.

4.2. Individual Views

The *scatterplot matrix view* gives an instant overview of a metamodel, but also supports sensitivity analysis. It is based on the common concept of a scatterplot matrix and thus contains a scatterplot for every pair of parameter–criterion combinations, therefore giving an overview of the distribution of values. Through the combination of zooming and filtering with the different capabilities of point sampling, the scatterplot matrix view is of key importance for local and global sensitivity analysis. Furthermore, it allows for navigating the data space by clicking the projections of individual samples.

As the second point visualization, we include the *parallel coordinates view* [Gan83]. It visualizes the same set of points as the scatterplot matrix view. Because of visualizing relations between neighboring axes, it enables users to analyze sensitivities of parameters and criteria. Additionally, it visualizes the common gradient trajectory for the used focal point to provide a more sophisticated means of local sensitivity analysis. To this end, a point set is generated by sampling the gradient trajectory uniformly over its length. Its visualization depicts the ranges covered on individual axes at a glance, while also revealing local interdependencies. It can be shown as overlay or stand-alone.

The *HyperSlice view* [vWvL93] gives users a comprehensive view on the multi-dimensional vicinity of focal points. It improves the overall understanding of the data, but also directly depicts potential optimizations. HyperSlice is a matrix of axis-aligned 1D and 2D slices through the parameter space. It consists of three parts: the upper triangular part, the lower triangular part, and the diagonal. The upper and lower parts each show all possible axis-aligned 2D slices through the respective focal point. The diagonal of the matrix shows all axis-aligned 1D slices through the focal point.

This layout implies that the upper and lower part display transposed views on the same data. Though this might seem redundant at first, it features several benefits for metamodel analysis: it enables users to emphasize different aspects of the data by applying different color mappings for the upper and lower triangular part. Alternatively, users can analyze two different criteria or two different focal points simultaneously, thus supporting comparisons and multi-criteria optimization. To further support the understanding of metamodels, the 2D slices are superimposed with additional information: projections of POI and the common gradient trajectory originating from the used focal point.

In contrast to the HyperSlice view, the *3D view* only shows a single 3D slice with user-defined axes. Thus, it does not give an overview over all dimensions, but instead fully depicts three parameters, thereby providing more context for users. To improve spatial understanding, the 3D view contains three crossed 2D slices of which the intersection point matches the projection of the focal point. Furthermore, we display a gradient trajectory through the focal point and POI.

Analyzing cutting processes, analysts often have an interest in gaining knowledge about the actual shape of the resulting cut kerf. To provide such information, the optional *cut shape view* illustrates the shape of the cut kerf produced by a cutting process. For the view to be present, the metamodel must provide the *cut depth* as parameter and the resulting *cut width* as criterion. Additionally, scaling information for both attributes must be provided. If these conditions are met, the cut shape view is shown below the HyperSlice view.

5. Parallel Metamodel Updates

In order to facilitate interactive exploration of large metamodels, it is crucial to provide fast updates for each of the CMV components discussed above. In this section, we propose different optimization strategies to achieve instant overview feedback, based on which the user can decide whether to wait for further refinement or to continue exploration with different parameters right away. This does not only support interactive exploration, but ultimately helps in arriving at better results faster.

The main computational load is generated by metamodel evaluations, either from probing the RBFN during slice updates or from gradient evaluation during trajectory tracing. For each evaluation, the contribution of all training points—the inner sums in Equations 1 and 4—has to be computed for the metamodel and if provided, for the limits metamodel. We deliberately decided to use a CPU-based parallelization approach for a number of reasons: first, it makes *memoSlice* available on platforms that do not feature dedicated GPUs; second, it allows us to integrate *memoSlice*—or parts thereof—into other application contexts, such as virtual production prototyping [PGK*14].

5.1. Asynchronous Parallel Computations

RBFN evaluations are embarrassingly parallel in nature at two levels. Each RBFN evaluation can be performed independently and the contributions of individual RBFs can be computed in parallel. We translate these observations into a layered, task-based parallelization scheme for the CPU. On the top level, a single task comprises the update of one 1D, 2D, or 3D slice, the computation of a gradient trajectory, the sampling of the common point set, and other small-scale tasks for certain application components.

The task for updating a slice is decomposed into sub-tasks, each of which providing several samples. Each of these tasks in turn spawns sub-tasks, which evaluate and sum up individual RBF-kernels, utilizing the parallel reduce pattern. Gradient trajectory tracing results in two sub-tasks for forward and backward tracing, respectively. In every tracing step, these tasks are decomposed in parallel RBF evaluations to compute ∇f, again, utilizing the parallel reduce pattern. The task for sampling the common point set is decomposed analogously to slice updates. Other components also directly probe metamodels for individual values, which is reflected by a corresponding small-scale task. For instance, the evaluation of criterion values of focal point previews is carried out asynchronously to avoid impact on the frame rate when analyzing large metamodels. *memoSlice* additionally uses other small-scale tasks for computations that might exceed the duration of a single frame, e.g., determining the focused point in scatterplots for direct discrete navigation.

Load-balancing across CPU cores and low-level task scheduling are handled by the Intel® Threading Building Blocks (TBB) runtime [Int18b]. For scheduling it primarily relies on a thread pool with task stealing for idling threads. The task engine runs asynchronously to the frame loop, hence not interfering with continuous rendering updates, but saturating all other available CPU cores. In *memoSlice* a custom task manager spawns high-level tasks whenever depictions need to be updated and updates are displayed as soon as they become available. In contrast, small-scale tasks are directly spawned by the components requiring their computations. The full task hierarchy is depicted in Fig. 2.

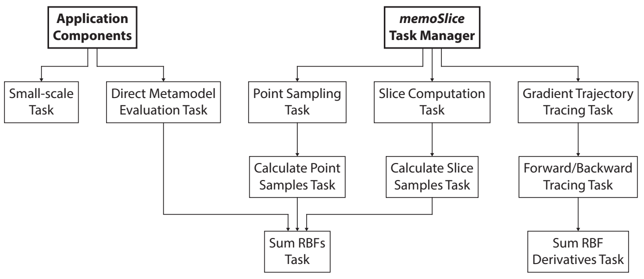
Application Components
memoSlice Task Manager
Small-scale Task
Direct Metamodel Evaluation Task
Point Sampling Task
Slice Computation Task
Gradient Trajectory Tracing Task
Calculate Point Samples Task
Calculate Slice Samples Task
Forward/Backward Tracing Task
Sum RBFs Task
Sum RBF Derivatives Task

Figure 2: *Task hierarchy of* memoSlice. *The task manager spawns top level tasks for visualization updates. Other application components issue small-scale tasks as needed. All tasks recursively spawn sub tasks, where feasible.*

5.2. Streaming Updates

Most metamodels feature smooth gradients over extended parts of the data domain. Hence, it is possible to gain a first overview of the data field based on coarse sampling. Precision adjustments require a more accurate depiction of the data. To reduce the perceived system latency and to update the visualizations as quickly as possible, we propose progressive computation of slice-based views. Slice updates start with a minimum resolution and are refined by interval bisection until a maximum resolution is reached (5 to 129 samples per axis; see Fig. 3). The progressive update scheme provides nearly instant overviews, with further refinements being streamed in upon availability. Since new samples are inserted between already calculated ones, this calculation scheme does not generate computational overhead in terms of metamodel evaluations.

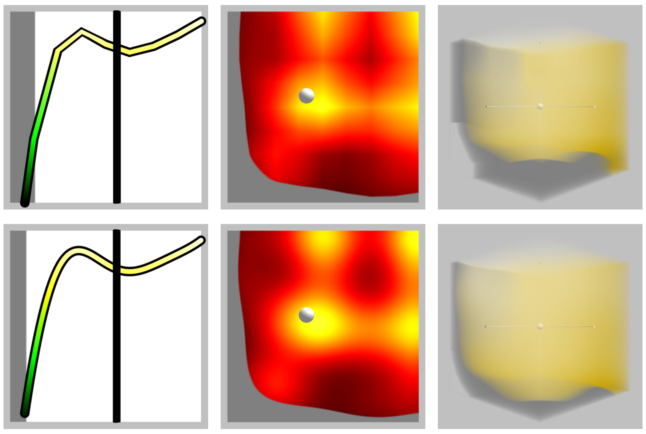

Figure 3: *1D, 2D, and 3D slices (l.t.r.) at minimum (top) and maximum (bottom) resolutions.*

We follow an alternative approach for updating the common point set. High numbers of points allow for a better understanding of relations in point visualizations. However, users can already draw first conclusions from small numbers of samples. Consequently, point visualizations are continuously updated as soon as new points are computed. To this end, exactly one top-level task for point sampling is active at a time per point set. The number of points to be computed by this task is chosen on an estimate so that new points are available in fixed time intervals (by default, $0.25\,s$). Whenever such a computation task terminates, the newly computed points are added to the common point set and the point visualizations are updated. If more points need to be sampled, a continuation task is spawned immediately.

5.3. Avoiding Redundant Computations

Update computations can be reduced significantly by exploiting the fact that multiple visualization primitives in *memoSlice* display different visual representations of the same data. For example, projections of the same gradient trajectory are overlayed on multiple tiles of the HyperSlice view. To avoid redundant computations, we label each computation by a unique key, which is derived from its configuration, e.g., a slicer is uniquely identified by its set of parameters, its criterion, focal point, and metamodel. Visualization primitives use this key to query the respective computations from registries and receive the corresponding data objects, which automatically notify the visualization about incoming updates. For instance, the upper-left and lower-right slice visualizations of the HyperSlice view in Fig. 1 would both request a slicer that creates slices along the *PulseDuration – ThicknessLevel* parameters of the limits metamodel and share the output data. The registries—at the moment one exists for each, slicers and gradient trajectories—store all computations that are currently used by at least one visualization primitive. The visualization primitives are responsible for transforming the data to match their configuration, e.g., transposing a slice as needed and applying color mapping.

Recomputations of slices, gradient trajectories, and the common point set are triggered by events, e.g., a change to the respective focal point or the adjustment of visual ranges. However, in many situations such recomputations are only partly needed or can fully be avoided. Regarding slices, a change to the focal point has an effect on the slice iff the change took place in dimensions that are not sliced through. Consequently, unaffected slices do not need to be updated. In contrast, changes to the visible range of a parameter only require slices to be recomputed if the changed parameter is part of the slice. To further reduce the perceived latency of such a change, we scale the textures used for displaying the slices until updated computations are ready.

Regarding gradient trajectories, these are always affected by focal point changes. In contrast, adjustments to the visible ranges only affect the visual representations of gradient trajectories. Consequently, such adjustments are handled by the respective visualization primitives within frame updates.

The need for recomputing the common point set depends on its current configuration. If uniform random distribution sampling is active, a change to the respective focal point causes recomputation iff changed parameters are currently locked. Since this sampling method is used most of the time and changes to locked parameters happen rarely, also the common point set rarely needs to be recomputed. In contrast, for sampling with normal distribution, any change to the respective focal point requires a recomputation. No matter which sampling method is used, changes to the visible range always require recomputation of the common point set. Narrowing the range requires to replace excluded points with newly created ones, while widening the range requires recomputation for the newly included areas.

5.4. Task Scheduling

An advantage of the task-based parallelization scheme is that task spawning can be prioritized in a user-centric fashion. By this, we aim at further reducing the perceived latency between interaction and visual feedback. Moreover, in the case of rapidly succeeding user interactions, we are able to cancel out-of-date tasks and issue new ones to account for the latest application state.

Task spawning in *memoSlice*' task manager is triggered by events like a change to the focal point, a switch of the displayed criterion, and the termination of an update computation. For an incoming event, we check for each individual slice and gradient trajectory, as well as the common point set whether the event invalidates its current state and thus demands an update of the respective calculation. Due to their low number of high-level tasks and potentially long-lasting computations, tasks for the computation of gradient trajectories and the common point set are spawned immediately when respective updates are required.

Progressive refinement of slices demands more sophisticated task scheduling. A top level task is issued for every new refinement iteration of a slice. Since TBB does not make guarantees for the order of task executions, spawning new refinement tasks whenever possible could result in unbalanced resolution levels. This would hinder the intent of providing an instant overview with refinements being streamed in on over time. To mitigate this issue, we develop a task scheduling scheme, based on the following observations.

First, we consider the complexity of slice computations. For the progressive refinement of slices, we define the *refinement level* ($l \in \mathbb{N}_+$) of any slice to be the current iteration in the refinement process, starting at $l = 1$. Interval bisection requires an odd number of samples per refinement level, so the number of samples per slice axis $n \in \mathbb{N}_+$ is given by $n = 2^l + 1$. The complexity of the calculation of a slice with dimensionality $d \in \mathbb{N}_+$ is $\mathcal{O}(n^d)$ or, when considering the refinement level, $\mathcal{O}(2^{ld})$. I.e., the calculation time for a slice computation iteration exponentially grows with both, the refinement level and the dimensionality. Consequently, the fastest user feedback can be achieved by first focusing on lower refinement levels and dimensionalities when scheduling slice computation tasks. However, to grant nearly-instant overview to users, it is still important to present slices of higher dimensionality as fast as possible. These observations lead to the following scheduling theme for slice computation tasks.

Upon an incoming task spawning event, we decide for each slice in the slice registry if the next refinement iteration is considered to be scheduled based on the following rules: first, if the final refinement iteration is already reached, no further iterations are needed. Otherwise, we perform two additional tests: first, the *within dimension* test. It succeeds if the next refinement level for the current slice is at maximum one level ahead of the lowest completed refinement level among all slices with the same dimensionality. This test guarantees, that all slices of the same dimensionality are refined in a balanced fashion, e.g., that all graphs displaying 1D slices have about the same resolution. If that test passed, we perform the *between dimensions* test. For 1D slices, this test always succeeds. For slices of higher dimensionality, this test succeeds, if the refinement level of the slice is less than the lowest completed refinement level of any unfinished slice with lower dimensionality. This way, we prioritize the refinement of slices of lower dimensionality, while allowing the other slices to catch up soon.

Another important observation regarding slice computations is that using different interpolators (see Sec. 3) and metamodels might result in varying computation times. To account for this, we group all slice computations by these attributes plus the used focal point, which is needed for a later step. In the previously described tests, only slices in the same group are compared to each other. Consequently, slicing levels within groups increase in a balanced fashion, while groups with shorter computation times are allowed to advance faster than those with longer computation times.

To finally decide, which refinement tasks are spawned, we further consider user interaction with the goal of minimizing perceived latency by prioritizing updates in the focus of user attention. To this end, we store the time stamp of the last user interaction for each parameter of each focal point. We choose the refinement task for the slice the user interacted with last for each of the previously discussed groups. If no other task from the respective group is currently running, the task manager finally spawns the chosen task.

The last limitation—only having a maximum of one top level task running per group of slice computations—further adds to the goal of reducing the perceived latency. This way, refinements in the focus of users' attention are computed first. Yet, the overall computation time for one refinement iteration over all slices is not higher compared to spawning all refinement tasks in parallel. This is due to our layered parallelization scheme, which draws benefit from concurrency by performing task parallelization on deeper layers.

6. Performance Analysis

To prove the applicability of our performance optimization approaches, we present two types of performance measurements of different complexity. First, we discuss the scalability of our metamodel evaluation optimization by analyzing micro measurements carried out on a state of the art multi-core workstation. Second, we illustrate the benefits of the combination of our individual optimization approaches by presenting measurements from a close-to-real-use lab setting that outlines average wait times for the calculation of individual visualization primitives in *memoSlice*.

6.1. Performance Micro Measurements

We present performance measurements, carried out on a high-end consumer-grade workstation featuring two Intel® Xeon® CPUs E5-2695 v3 with a clock frequency of 2.30 GHz. Each of the CPUs has 14 physical / 28 logical cores, resulting in 28 physical / 56 logcial cores for the test system. It is equipped with 512 GB of RAM.

6.1.1. Procedure

We analyzed the performance and scalability of our approach for the independent variables *number of cores* (c), *number of training points* (n_t), *number of input dimensions* (n_d), *RBF kernel type* (ϕ), and *parallelization hierarchy depth* (h). For each measurement, we created a synthetic data set with randomly distributed training points and random RBF weights. We then performed 100,000 metamodel evaluations at random locations in the data space and measured the *time* for the evaluations (t). Depending on t, we calculated the *sampling rate* (r) as $r = 100,000/t$. Additionally, we calculated the *parallel efficiency* (e_c) depending on c as $e_c = \frac{r_c}{r_1 c}$ where r_c is the sampling rate with c cores and r_1 is the sampling rate with 1 core.

In our measurements, we varied c by initializing the thread pool of the TBB task engine with c threads. If not stated differently, $c = 28$, which corresponds to the number of physical CPU cores of our test system. With $c > 28$, logical cores are used in addition to physical ones. The levels measured for n_t are $10^i : i \in [1,\dots,5] \subset \mathbb{N}$. From these, 10^2–10^4 represent magnitudes of training point amounts from production metamodels. We included the other amounts to cover boundary conditions. If not stated differently, $n_t = 10^4$, which corresponds to large production metamodels. Regarding n_d, we performed measurements for 4, 6, and 8 input dimensions, which are regularly occurring values for metamodels. If not stated differently, $n_d = 6$, which is most widely used. We measured the performance of the three kernel types presented in Sec. 3: ϕ_m, ϕ_g, and ϕ_l. If not stated differently, $\phi = \phi_m$, since it is the default kernel for encoding criteria in production metamodels. To simulate the layered parallelization scheme of *memoSlice*, we performed measurements with two different levels of h. In the flat hierarchy setting F, all 100,000 metamodel evaluations were carried out serially, i.e., with only the evaluation of the RBFN being computed in parallel. In contrast, in the deep hierarchy setting D, the metamodel evaluations were carried out in parallel, thus providing two layers of parallelization and mimicking the computation of slices and the common point set. If not stated differently, $h = D$, since this type of evaluation is carried out most.

6.1.2. Results

The highest sampling rate of $r = 3.0 \times 10^6 \frac{1}{s}$ was achieved with $n_t = 10$ and $c = 56$. Since these conditions do probably not resemble a situation that occurs in practice, we consider this value to be a theoretical upper boundary for the sampling rate achievable with our test system. Fig. 4 reveals that c has a negligible impact on r for metamodels with $n_t = 10$. However, for metamodels with $n_t = 10^2$, $c = 8$ imposes a considerable performance benefit, with a close-to-linear correlation of n_t and r for metamodels with $n_t \geq 10^2$. For metamodels with $n_t \geq 10^3$, $c = 28$ imposes a further performance increase, with a close-to-linear correlation of n_t and r.

We found that the parallelization hierarchy depth h has a strong impact on the parallel efficiency e_c, which is illustrated in Fig. 5. For $h = F$ and $n_t = 10^3$, e_c already falls below 50 % when using 4 cores and it falls below 10 % when using 20 cores. For larger metamodels ($n_t = 10^4$), e_c is still slightly above 80 % with 4 cores, while falling below 50 % with 11 cores. In contrast, $h = D$ yields clearly higher levels for e_c: for medium-sized metamodels ($n_t =$

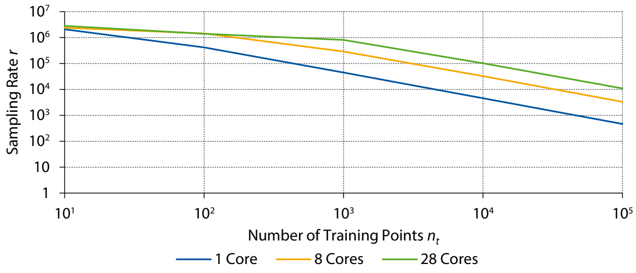

Figure 4: *Sampling rate of metamodel evaluations for metamodels of varying size.*

10^3), e_c is above 80 % when using less than 10 cores, while it is still above 65 % for 28 cores. For large metamodels ($n_t = 10^4$), e_c has overall high values, with results around 80 % even when using 28 cores. When additionally using logical cores, e_c for $h = D$ stagnates faster, but still yields considerable speed gains with $e_c = 34\,\%$ ($n_t = 10^3$) and $e_c = 42\,\%$ ($n_t = 10^4$) for $c = 56$.

These measurements illustrate that $h = D$, i.e., using a layered parallelization approach, implies considerably larger speed gains with a rising number of CPU cores compared to $h = F$. They also show that the evaluation of larger metamodels benefits more from parallelization than the evaluation of smaller ones. In *memoSlice*, tasks with both, flat and deep parallelization hierarchies are carried out. However, as illustrated in Fig. 2, only few metamodel evaluations are carried out with a flat hierarchy, i.e., individual evaluations by application components. In contrast, the largest amount, i.e., slice and point sampling tasks, are carried out with a deep hierarchy. Considering the performance measurements, we deem this approach suitable for practical use. Additionally, we recommend the use of simultaneous multithreading for running *memoSlice* whenever available, since the use of logical cores imposes considerable speed gains, even if e_c stagnates faster than with solely using physical cores.

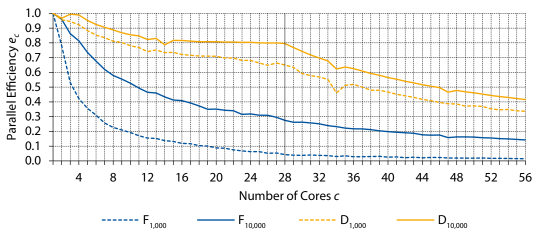

Figure 5: *Parallel efficiency of metamodel evaluations for flat and deep parallelization hierarchies.*

Considering the different RBF kernels, our measurements show that ϕ_l is evaluated fastest, followed by ϕ_m and ϕ_g (see Fig. 6). For small metamodels ($n_t = 10^2$), ϕ accounts for approximately 10 % performance impact ($r_{\phi_m}/r_{\phi_l} = 0.91$, $r_{\phi_g}/r_{\phi_l} = 0.89$). On the contrary, for metamodels with $n_t \geq 10^3$, the sampling rate with ϕ_m is about 90 % of the sampling rate with ϕ_l ($r_{\phi_m}/r_{\phi_l} = 0.91$), while it is about 50 % with ϕ_g ($r_{\phi_g}/r_{\phi_l} = 0.50$). These values are means over the sampling rates for $n_t = 10^3$, $n_t = 10^4$, and $n_t = 10^5$.

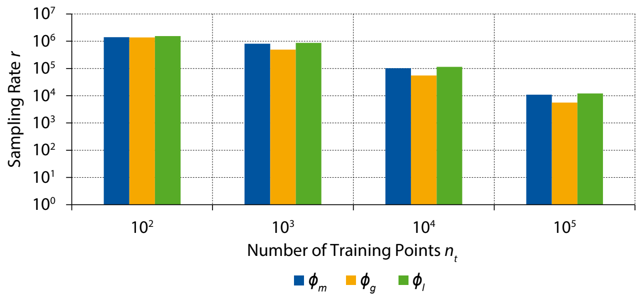

Figure 6: *Sampling rates with the different RBF kernels for varying numbers of training points.*

Regarding the number of input dimensions, an increase in n_d causes a decrease on r (see Fig. 7). This effect gets pronounced with a rising number of training points. For metamodels with $n_t = 10^3$ and $n_d = 8$, r resides at approximately 72 % compared to metamodels with the same number of training points and $n_d = 4$ ($r_{n_d=8}/r_{n_d=4} = 0.718$). For metamodels with $n_t = 10^4$ this ratio is approximately 65 % ($r_{n_d=8}/r_{n_d=4} = 0.652$), while it is approximately 62 % with $n_t = 10^5$ ($r_{n_d=8}/r_{n_d=4} = 0.615$).

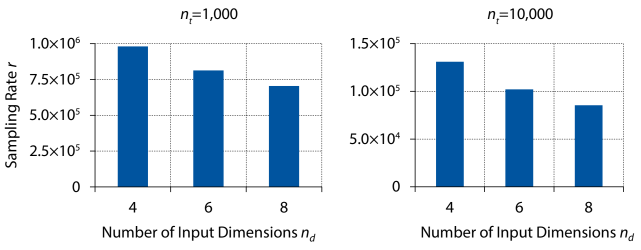

Figure 7: *Sampling rates for different numbers of input dimensions.*

To get further hints on optimization potential, we analyzed our test program with Intel® VTune [Int18c], a widely-accepted performance analysis tool. The results show an ideal *effective time by utilization* for RBFN evaluation. Furthermore, its *clockticks per instructions retired (CPI) rate* is 0.62. As a reference, Intel architects set up a value of 0.75 and below as "good" [Int18a]. These results indicate that no or little bottlenecks regarding I/O or memory bandwidth exist, i.e., that the efficiency is mainly CPU-bound.

6.2. High-level Performance Measurements

To illustrate the benefits of combining our methods to achieve high levels of responsiveness, we present high-level performance measurements that aim at closely resembling real metamodel analysis sessions. The measurements were carried out on a high-performance consumer-grade workstation featuring two Intel® Xeon® CPUs E5-2620 v4 with a clock frequency of 2.10 GHz. Each of the CPUs has 8 physical / 16 logical cores, resulting in 16 physical / 32 logical cores for the system. It is equipped with 32 GB of RAM.

6.2.1. Procedure

Our goal for the high-level performance measurements is to create a controlled lab setting that resembles real analysis sessions as closely as possible. We augmented *memoSlice* with methods for artificial user input and time measurement. We measured the times to compute the individual slicing iterations for all 1D, 2D, and 3D slices, the time to compute the individual iterations for the common point set, and the computation time for the gradient trajectories. We performed these measurements for three production metamodels of different sizes: small (M_s: 6 parameters, 1,160 training points for the criterion metamodel, 1,727 training points for the limits metamodel), medium (M_m: 5 parameters, 8,949 training points for the criterion metamodel, 9,990 training points for the limits metamodel), and large (M_l: 5 parameters, 20,237 training points for the criterion metamodel, 9,990 training points for the limits metamodel).

To conduct the measurements, *memoSlice* was configured to display the first criterion of the metamodel in the upper part of the HyperSlice view and the 3D view. The lower part of the HyperSlice view displayed the training point density. Uniform sampling was used to generate the common point set. The application always displayed the full data space.

To mimic user input, we added a frame-loop callback that changed the focal point after all computations had reached their maximum levels. These were 100,000 points for the common point set, 129 samples per axis for 1D and 2D slices, and 65 samples per axis for 3D slices. Further, all gradient trajectories were to be computed, i.e., 2 in total, always one for the first criterion and the training point density.

A change to the focal point would usually not trigger recomputation of the common point set (see Sec. 5.3) in the described test configuration. To account for this, we disabled this redundancy avoidance feature, to gather data on point computation times.

Considering standard workflows of real analysis sessions, recomputation for all visualization primitives as simulated in our measurements rarely occurs. However, such types of interaction cause the longest wait times for users in real analysis sessions, which is why we chose such a setting to illustrate the upper boundaries of user wait times.

6.2.2. Results

For each metamodel we performed 500 successive measurements. We present the mean computation times for the following variables: 1D, 2D, and 3D slices for the first criterion of the metamodel and the limits metamodel ($1D_C$, $1D_L$, $2D_C$, $2D_L$, $3D_C$, and $3D_L$), gradient trajectories for the first criterion and the training point density of the metamodel (GT_C and GT_D), and the common point set (P). These variables represent the individual scheduling groups of our task scheduling scheme (see Sec. 5.4). For slices, the iterative refinement resolutions included in our measurements are 9, 17, 33, 65, and 129 samples per axis for 1D slices; 5, 9, 17, 33, 65, and 129 samples per axis for 2D slices; 5, 9, 17, 33, and 65 samples per axis for 3D slices. For gradient trajectories, we present the mean computation time and the average number of integration steps. For the computation of the common point set, our measurements include the mean times of updates at which at least 10^2, 10^3, 10^4, and 10^5 points were sampled, respectively. We classify individual measurements whose values deviate more than three times the standard deviation from the mean value as outliers and exclude them from our analysis. This accounts for 2.2 % of the measurements for M_l, 2.1 % for M_m, and 1.1 % for M_s. We discuss these at the end of our analysis.

Figure 8 depicts an overview of the results. It illustrates the average computation times for the individual variables on multiple time lines, where each dot represents the termination of a new refinement iteration.

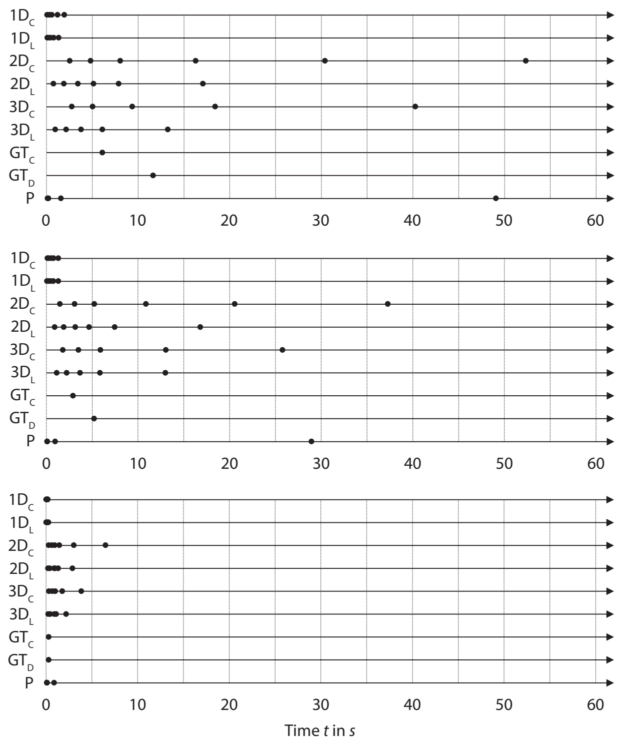

Figure 8: *Mean computation times for updates of the individual visualization primitives for M_l, M_m, and M_s (t.t.b.).*

Regarding slices, for all metamodels, several refinement iterations for 1D slices were available within the first second after a focal point change. The longest computation time for the first iteration of 2D slices was encountered for the criteria of M_l with $2.6\,s$, followed by the criteria of M_m $1.5\,s$. For both metamodels, the first iteration of the limits metamodel was computed in below $1\,s$. For M_s three iterations (17^2 samples) were computed in below $1\,s$ for both, the criteria and the working limits. Computation times for the first iteration of 3D slices were slightly higher for all metamodels: $2.8\,s$ and $1.0\,s$ for criteria and working limits of M_l, respectively, and $1.8\,s$ and $1.1\,s$ for criteria and working limits of M_m, respectively. Three iterations (17^3 samples) were finished within the first second for both metamodels of M_s.

Regarding the computation of the common point set, one second after a focal point change, more than 1,000 points were available for M_l, while it lasted $49.1\,s$ on until 100,000 points were ready. For M_m it were more than 10,000 after the first second while the final iteration of 100,000 points was computed after $29.0\,s$. For M_s, after the first second, the final iteration of 100,000 points was already computed.

The longest average computation times for gradient trajectories were encountered for M_l, with $6.1\,s$ and $11.7\,s$ for the first criterion and the training point density, respectively. The respective average numbers of integration steps were 304 and 475. For M_m, the average computation times were $2.9\,s$ and $5.2\,s$ with average respective numbers of integration steps of 273 and 369. For M_s, the gradient trajectories for both criteria were available after $0.3\,s$ with average respective numbers of integration steps of 106 and 76.

6.3. Discussion

The presented micro measurements give valuable hints on the efficiency of *memoSlice*. We could demonstrate that the evaluation of production-size metamodel can be performed fast enough on modern hardware to theoretically enable users to perform interactive exploration. In this context, our layered parallelization scheme is a key-enabler and we showed that we achieve a reasonable parallel efficiency, which is particularly important for the analysis of larger metamodels. Additionally, we illustrated the impact of several variables of metamodels and can conclude that, among these, the number of training points has the strongest effect on the sampling rate. Furthermore, the performance analysis with VTune showed that the current implementation is mainly CPU-bound, indicating a good utilization of the underlying hardware with little room for further improvements.

To complement these conclusions the results of our high-level measurements confirm that our acceleration methods make *memoSlice* suitable for interactive visualization of metamodels. User feedback in form of 1D slices was nearly instantly available for all metamodels, while longer wait times were present for 2D and 3D slices for the larger metamodels. With always less than $3\,s$, we still deem these wait times acceptable for interactive analysis. For the medium-sized metamodel, first iterations of 2D and 3D slices were available within the first second after interaction. Additionally, a reasonable size of the common point set was available the first second after interaction for all metamodels.

Regarding gradient trajectories, the average computation times varied strongly, not only over the tested metamodels but also over the criteria. This shows that the computation time of such trajectories does not only depend on the actual size of a metamodel, but also on the number of integration steps needed for their computation. We attribute the varying number of integration steps to the different topologies of the analyzed criteria. In general, the wait times for gradient trajectories are acceptable for large metamodels while for medium-sized metamodels, they are negligible.

Over all of our measurements we identified between $1.1\,\%$ and $2.2\,\%$ of the individual runs to contain outliers. Reproducing situations that lead to such long computations for individual task groups, we found that the computation of a gradient trajectory or the common point set can saturate most or even all of the available CPU cores. We attribute this behavior to a high number of low-level tasks being distributed over the majority of cores via TBB's task stealing policy. To avoid such a situation, the high-level task scheduling scheme of *memoSlice* should be refined in a way that guarantees that all computations needed to provide an overview are finished before tasks that can saturate all cores are issued. Nonetheless, we do not see the relatively rare occurrence of outliers to be critical. In such cases, successive user interaction aborts ongoing computations and starts rescheduling of tasks.

In general, it should be taken into account that, due to the chosen test setup, the measured wait times represent upper boundaries. Consequently, shorter wait times are to be expected in practical use. We can conclude that the response times of *memoSlice* for short- to medium-sized metamodels are negligible, while short wait times are to be anticipated for large metamodels. Thus, *memoSlice* can be considered to be a suitable solution for the interactive visual analysis of metamodels with regard to performance.

7. Conclusion

In this paper, we have presented *memoSlice*, a visualization appication that targets the interactive exploration of multi-dimensional metamodels. As stated in the introduction, the development faced two challenges: making the data accessible and providing interactive user feedback. To this end, we have proposed a CMV design that integrates a number of carefully designed views in order to grant users a holistic understanding of the local data space. As a downside, this design critically depends on the fast evaluation of metamodels, because it requires a large number of such evaluations, particularly after user interaction. In order to reduce response times, we have devised several acceleration strategies, most importantly, our task-based parallelization approach. We have presented evaluations of our acceleration approaches via micro and high-level measurements. Considering the micro measurements, we conclude that our evaluation scheme, in particular, grants considerable performance improvements and that it scales well with a rising number of CPU cores. Additionally, our high-level measurements demonstrate acceptable response times for interactive exploration of production-size metamodels in *memoSlice*.

We would like to note that until now we performed all measurements in artificial lab settings. We currently plan to roll out *memoSlice* for a large user population and gather performance data from daily use. By combining such data with respective user feedback, chances for additional improvements might be discovered. Nonetheless, the performance measurements carried out so far already show that the employed methods make *memoSlice* responsive enough to analyze metamodels via interactive visualization.

References

[AKES15] Al Khawli T., Eppelt U., Schulz W.: Advanced Metamodeling Techniques Applied to Multi-dimensional Applications with Piecewise Responses. In *Machine Learning, Optimization, and Big Data, First International Workshop, MOD 2015* (2015), vol. 9432, Springer International Publishing. 2

[Al 15] Al Khawli T.: *A metamodeling approach towards virtual production intelligence*. PhD thesis, RWTH Aachen University, München: Verlag Dr. Hut – Ingenieurwissenschaften, 2015. 2

[BHEP04] Bremer P.-T., Hamann B., Edelsbrunner H., Pascucci V.: A Topological Hierarchy for Functions on Triangulated Surfaces. *IEEE Trans. Vis. Comput. Graphics 10*, 4 (2004), 385–396. 2

[BMPM12] Booshehrian M., Möller T., Peterman R. M., Munzner T.: Vismon: Facilitating Analysis of Trade-Offs, Uncertainty, and Sensitivity In Fisheries Management Decision Making. *Comput. Graph. Forum 31*, 3 (2012), 1235–1244. 2

[BPFG11] Berger W., Piringer H., Filzmoser P., Gröller E.: Uncertainty-Aware Exploration of Continuous Parameter Spaces Using Multivariate Prediction. *Comput. Graph. Forum 30*, 3 (2011), 911–920. 2

[BWP*10] Bremer P.-T., Weber G. H., Pascucci V., Day M., Bell J. B.: Analyzing and Tracking Burning Structures in Lean Premixed Hydrogen Flames. *IEEE Trans. Vis. Comput. Graphics 16*, 2 (2010), 248–260. 2

[CSvdP10] Carr H., Snoeyink J., van de Panne M.: Flexible Isosurfaces: Simplifying and Displaying Scalar Topology Using the Contour Tree. *Computational Geometry: Theory and Applications 43*, 1 (2010), 42–58. 2

[EDF08] Elmqvist N., Dragicevic P., Fekete J.: Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation. *IEEE Trans. Vis. Comput. Graphics 14* (2008), 1148–1539. 1

[EHNP03] Edelsbrunner H., Harer J., Natarajan V., Pascucci V.: Morse-Smale Complexes for Piecewise Linear 3-Manifolds. In *Proc. of the Annual Symposium on Computational Geometry* (2003), pp. 361–370. 2

[Gan83] Gannet H.: General Summary Showing the Rank of States by Ratios 1880. In *Scribner's statistical atlas of the United States showing by graphic methods their present condition and their political, social and industrial development*, Hewes F. W., (Ed.). United States. Census Office, 1883, p. 151. 4

[GBHP08] Gyulassy A., Bremer P.-T., Hamann B., Pascucci V.: A Practical Approach to Morse-Smale Complex Computation: Scalability and Generality. *IEEE Trans. Vis. Comput. Graphics 14*, 6 (2008), 1619–1626. 2

[GBPW10] Gerber S., Bremer P.-T., Pascucci V., Whitaker R.: Visual Exploration of High Dimensional Scalar Functions. *IEEE Trans. Vis. Comput. Graphics 16*, 6 (2010), 1271–1280. 2

[GKK*12] Gyulassy A., Kotava N., Kim M., Hansen C. D., Hagen H., Pascucci V.: Direct Feature Visualization Using Morse-Smale Complexes. *IEEE Trans. Vis. Comput. Graphics 18*, 9 (2012), 1549–1562. 2

[Int18a] Intel®: Clockticks per Instructions Retired (CPI) . https://software.intel.com/en-us/node/544403/, 2018. Accessed: 2018-03-04. 8

[Int18b] Intel®: Intel® Threading Building Blocks (Intel® TBB). https://www.threadingbuildingblocks.org/, 2018. Accessed: 2018-03-04. 5

[Int18c] Intel®: Intel® VTune™ Amplifier 2017. https://software.intel.com/en-us/intel-vtune-amplifier-xe/, 2018. Accessed: 2018-03-04. 8

[KH13] Kehrer J., Hauser H.: Visualization and Visual Analysis of Multifaceted Scientific Data: A Survey. *IEEE Trans. Vis. Comput. Graphics 19*, 3 (2013), 495–513. 1

[Kut01] Kutta M. W.: Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeitschrift für Mathematik und Physik 46* (1901), 435–453. 3

[Max70] Maxwell J. C.: On Hills and Dales. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science 40* (1870), 421–427. 2

[Mil63] Milnor J. W.: *Morse Theory*. Princeton University Press, Princeton, N.J., 1963. 2

[MLP*10] McLoughlin T., Laramee R. S., Peikert R., Post F. H., Chen M.: Over Two Decades of Integration-Based, Geometric Flow Visualization. *Comput. Graph. Forum 29*, 6 (2010), 1807–1829. 2

[OHJS10] Oesterling P., Heine C., Janicke H., Scheuermann G.: Visual Analysis of High Dimensional Point Clouds Using Topological Landscapes. In *Proc. ofthe IEEE Pacific Visualization Symposium* (2010), pp. 113–120. 2

[Orr96] Orr M.: *Introduction to Radial Basis Function Networks*. Tech. rep., Centre for Cognitive Science, Edinburgh University, 1996. 2

[PBK10] Piringer H., Berger W., Krasser J.: HyperMoVal: Interactive Visual Validation of Regression Models for Real-Time Simulation. *Comput. Graph. Forum 29*, 3 (2010), 983–992. 2, 3

[PGK*14] Pick S., Gebhardt S., Kreisköther K., Reinhard R., Voet H., Büscher C., Kuhlen T.: Advanced Virtual Reality and Visualization Support for Factory Layout Planning. In *Proc. of the Conference Entwerfen Entwickeln Erleben* (2014), TUDpress. 5

[PSBM07] Pascucci V., Scorzelli G., Bremer P.-T., Mascarenhas A.: Robust On-line Computation of Reeb Graphs: Simplicity and Speed. *ACM Trans. Graphics 26*, 3 (2007), 58. 2

[Rob07] Roberts J. C.: State of the Art: Coordinated Multiple Views in Exploratory Visualization. In *Proc. ofthe International Conf. on Coordinated and Multiple Views in Exploratory Visualization* (2007), pp. 61–71. 2

[SHB*14] Sedlmair M., Heinzl C., Bruckner S., Piringer H., Moller T.: Visual Parameter Space Analysis: A Conceptual Framework. *IEEE Trans. Vis. Comput. Graphics 20*, 12 (2014), 2161–2170. 1

[Sma61] Smale S.: On Gradient Dynamical Systems. *Annals of Mathematics 74*, 1 (1961), 199–206. 2

[TSDS96] Tweedie L., Spence R., Dawkes H., Su H.: Externalising abstract mathematical models. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems* (1996), ACM, pp. 406–412. 3

[TWSM*11] Torsney-Weir T., Saad A., Möller T., Weber B., Hege H.-C., Verbavatz J.-M., Bergner S.: Tuner: principled parameter finding for image segmentation algorithms using visual response surface exploration. *IEEE Trans. Vis. Comput. Graphics 17*, 12 (2011), 1892–901. 2

[vWvL93] van Wijk J. J., van Liere R.: HyperSlice: visualization of scalar functions of many variables. In *Proc. of the 4th Conf. on Visualization '93* (1993), pp. 119–125. 1, 3, 4

[WDC*07] Weber G. H., Dillard S. E., Carr H., Pascucci V., Hamann B.: Topology-Controlled Volume Rendering. *IEEE Trans. Vis. Comput. Graphics 13*, 2 (2007), 330–341. 2