Computers & Graphics 109 (2022) 111–120

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

GPU accelerated scalable parallel coordinates plots

Josef Stumpfegger [a,*], Kevin Höhlein [a], George Craig [b], Rüdiger Westermann [a]

[a] Technical University of Munich, Boltzmannstraße 3, Garching, 85748, Germany
[b] Ludwig-Maximilians Universität München, Theresienstr. 37, 80333 München, Germany

ARTICLE INFO

Article history:
Received 10 August 2022
Received in revised form 24 October 2022
Accepted 28 October 2022
Available online 3 November 2022

Keywords:
GPU scattering
Parallel coordinates
Ensemble analysis

ABSTRACT

Parallel coordinates are a powerful technique to visually analyze multi-parameter data, i.e., sets of datapoints with potentially many associated parameter values per datapoint. When these sets are large, line rendering becomes a severe performance bottleneck, and since many lines fall into the same pixel the numerical precision of the color buffer is quickly reached. We propose a scalable GPU realization of parallel coordinates building upon 2D pairwise attribute bins, to significantly reduce the number of lines to be rendered. Our approach comprises a GPU compute pipeline that combines shader-based scattering with atomic increment operations to efficiently count how often a line is drawn. These counts are then used to draw all pairwise sub-plots in the parallel coordinates plot, by analytically calculating the opacity for each count and rendering a line with end points determined by the 2D coordinates of the bin. In this way, framebuffer precision issues that are paramount in classical approaches can be overcome. We demonstrate the efficiency of the proposed realization for visualizing a weather forecast ensemble comprising 2.7 billion datapoints, each carrying 7 prognostic floating-point variables like temperature, precipitation and pressure, plus spatial and simulation input variables. We compare our pipeline to a rasterization-based approach regarding performance, and demonstrate interactive brushing at 4 s per frame at full HD viewport resolution.

1. Introduction

Parallel Coordinates Plots (PCPs) are a powerful technique to visually analyze large sets of datapoints with multiple associated parameters [1]. While such plots can be realized efficiently on GPUs for some millions of datapoints, as this number goes into the tens or even hundreds of millions, the line rendering capabilities of even the strongest GPUs become a severe bottleneck. For instance, Kumpf et al. [2] demonstrate the use of PCPs for the analysis of meteorological ensembles and report interactive update rates of roughly 0.5 s for 2.5 million datapoints with 12 parameters each, with an optimized GPU solution. To overcome rendering issues of PCPs, Novotny and Hauser [3] introduce binned parallel coordinates, which use pairwise attribute-scatterplots (i.e., 2D histograms) that are precomputed on the CPU, to count how many datapoints draw the same line. For the analysis of the Hurricane Isabel dataset, Blaas et al. [4] build up on this work and include a data compression layer to support efficient disk-to-CPU streaming.

Others have proposed hierarchical aggregation in combination with precomputed and focus + context interactions to decrease the number of rendered primitives and reduce visual clutter due to many overlapping lines [6,7]. While it has been shown that such approaches can achieve high performance on parallel computing systems, they result in a significant increase of the required memory due the increase of the number of states of the visualization. As reported, even on two execution nodes with 12 cores and 31 GB memory each, multiple seconds are required to perform interactions such as selection and refinement for 200 million datapoints.

In our work, we consider a so-called Grand Ensemble comprising roughly 2.7 billion datapoints, each storing seven prognostic floating-point parameters like temperature, precipitation and pressure, along with three spatial variables and an additional 180 state simulation input variable (See Fig. 1.). For a single time step, this amounts to 56 GB of memory. Grand Ensembles are becoming an invaluable tool for assessing and quantifying the uncertainty in numerical simulations. An ensemble of simulations, for example weather forecasts, include members that sample different sources of uncertainty, which could include different initial conditions, different physical assumptions, and even different numerical approximations. A Grand Ensemble collects the data produced by a number of component ensembles, which have often been created independently and with different applications in mind, and treats it as a single dataset that can be analyzed as a whole.

As of today, beyond offline statistical approaches there is no support for an interactive explorative analysis of a Grand Ensemble on desktop PCs. When PCPs are generated in the classical way,

This article has been certified as Replicable by the Graphics Replicability Stamp Initiative: http://www.replicabilitystamp.org.
* Corresponding author.
E-mail address: ga87tux@mytum.de (J. Stumpfegger).

https://doi.org/10.1016/j.cag.2022.10.008

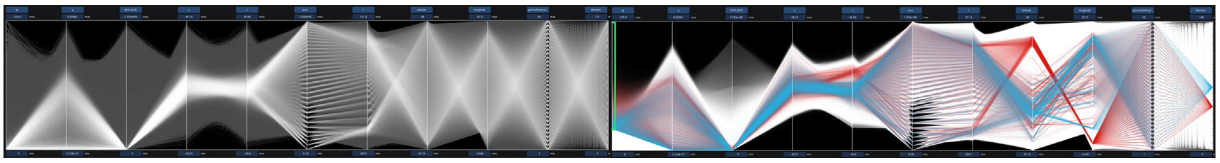

Fig. 1. Parallel Coordinates Plot (PCP) of one time step of a meteorological Grand Ensemble [5] comprising 2.7 billion datapoints with 11 parameters. The PCP is generated in roughly four seconds on a desktop PC with a single mid-range GPU. Left: All datapoints render their polylines with very low opacity of 10^{-7}. Lines drawn by less datapoints than a selected threshold are rendered with constant color as background. Right: Two ensemble members are emphasized by drawing their polylines in red and blue on top of all other polylines (rendered in white with medium opacity of 10^{-4}).

in our scenario about 2.7×10 billion lines (one line for each of the 10 variable pairs corresponding to the selected ordering of the PCP axes) need to be rendered in every frame. Notably, this takes multiple minutes on a high-range GPU. Rendering that many lines causes another problem that is related to the precision of the color buffer. Since lines falling into the same pixel are blended using either additive blending or α-blending, the numerical precision of the framebuffer is quickly reached and information is lost.

Contribution: We propose a scalable GPU realization of PCPs to enable an interactive visual analysis of large multi-parameter datasets on high-resolution displays. Scalability in the number of datapoints is achieved via binned parallel coordinates [3] that grow with the viewport resolution but not with the number of datapoints. This viewport-dependent data structure is generated instantly on the GPU and does not require any pre-baked hierarchical representation in which datapoints are aggregated. It encodes all possible visible lines in parallel coordinates by the vertical pixel coordinates of their end points on adjacent parameter axes. Then, computing the opacity of visible lines reduces to counting how often a coordinate pair occurs in all datapoints.

For building the data structure we propose an approach that is specifically tailored to the capabilities of recent mid- and high-range GPUs. It exploits the compute capabilities of GPUs to generate the data structure via parallel scatter and atomic increment operations. Scattering is performed into multiple render targets, i.e. one for each adjacent parameter pair in the PCP. The resolution of the render targets depends only on the height of the parameter axes in the pixel raster. For rendering the final lines between the PCP axes, a GPU line renderer reads the information from the render targets and converts it into a set of lines in a geometry shader.

On a single node CPU equipped with an Nvidia RTX 3090 GPU with 24 GB on-chip memory, the PCP of the Grand Ensemble can be generated in roughly four seconds on a 1920×1080 viewport, including streaming parts of the data from CPU RAM. If not all datapoints can be stored in CPU RAM, they need to be streamed from disk. To reduce the bandwidth requirements in such a situation, we provide the option to compress the parameter fields in a preprocess and generate the PCP from the compressed representation on the GPU. To distinguish between different members in the Grand Ensemble, the datapoints of selected members are binned separately, and their lines can then be rendered in different colors and specific orders to emphasize the ensemble composition. To analyze certain parameter intervals, the user can disable subsets of datapoints by brushing on the parameter axes.

Our specific technical contributions are:

- A GPU compute shader-based approach that instantly generates pairwise attribute-scatterplots using parallel scatter and atomic increment operations.
- The combination of pairwise attribute-scatterplots with analytical blending to accurately handle transparency.
- The realization of priority rendering via atomic minimum operations, which avoids sorting the datapoints.

Our PCP realization is implemented using the Vulkan Graphics API. The code is made publicly available under the Apache 2.0 license, and published at [8]. Since we cannot make the Grand Ensemble dataset publicly available, a synthetic dataset at the same size is provided. The current program has an importer for data in network common data format (NetCDF), as commonly used in atmospheric science workflows, as well as for comma separated file formats (CSV).

2. Related work

Multi-variate ensemble visualization. A number of different mapping techniques exist, which target the visualization of multivariate data, including axis-based methods, glyph encodings, as well as pixel-oriented and hierarchical approaches [9]. Scientific ensemble data, such as weather forecast ensembles, possess additional complexity due to the presence of spatial and temporal dependence in the data, as well as the uncertainty as conveyed in the ensemble dimension [10,11].

Wang et al. [12], Kumpf et al. [13] and Kumpf et al. [2] have shown that parallel coordinate plots (PCPs) are a promising tool for the comparison of multi-parameter distributions within a single and between different ensemble members. In PCPs [1,14], data dimensions are visualized as linear coordinate axes, which are arranged parallel to one another, and datapoints are represented as coherent lines crossing these axes. PCPs offer simple interaction possibilities for data selection, such as brushing [15], as well as good scalability to high data dimensions, which is an advantage over alternative visualization methods, such as scatter-plot matrices or radar charts [16,17]. With a focus on multi-parameter ensemble visualization, Kumpf et al. [2] present an optimized GPU-based visual analysis tool, which allows users to explore spatio-temporal ensemble datasets at interactive frame rates and with direct linking to secondary visualization tools such as direct volume rendering or violin charts for statistical distribution visualization.

Scalable parallel coordinate plots. Besides methods to improve the information content and visual quality of PCPs, such as optimal axis layouting, e.g. [18–22], curved polylines, e.g. [23–28], priority rendering [29], or slope-adaptive line rendering [30], another research direction has been towards scalable PCPs that can deal with large numbers of datapoints.

Most of these methods rely on rendering aggregated data representations instead of the raw data itself. For a review of methods, see Heinrich and Weiskopf [31]. Moustafa [32] describe scalability methods which rely on computing and visualizing summary statistics, which can be rendered faster and with less visual clutter than the raw data. Palmas et al. [33] cluster datapoints axis-wise and visualize connections between clusters through polygonal strips to reduce overplotting. McDonnell and Mueller

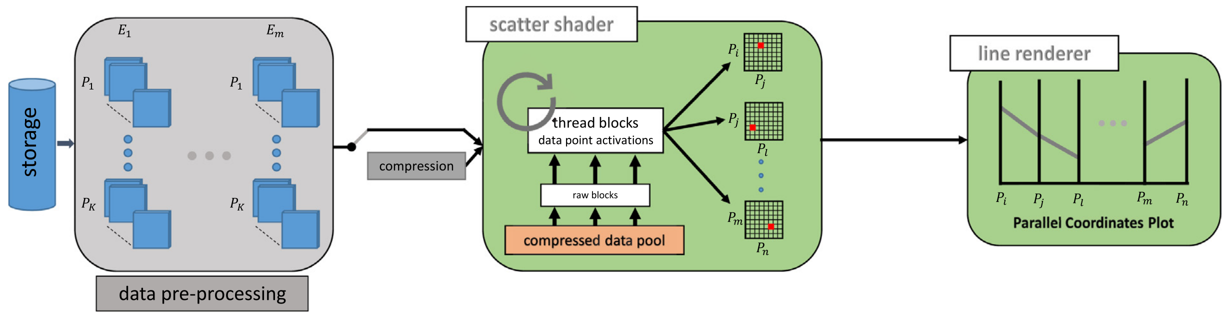

Fig. 2. Method overview. Green and gray backgrounds indicate GPU and CPU execution, respectively. Orange background indicates optional stages and data representations. In a pre-process, the parameter fields are read and structured, optionally compressed, and uploaded to the GPU. On the GPU, the data is read (requiring decompression if stored compressed), and scattered in a compute shader to multiple render targets to count the number of datapoints drawing the same line. The GPU line renderer interprets and renders the content of the generated buffers as lines. For the sake of clarity, we illustrate the case where all datapoints are scattered into the same set of render targets, regardless of the ensemble they come from.

[25] combine edge bundling and clustering of datapoints and render aggregate data representations with opacity and shading effects to distinguish between different aggregates. Alternative bundling techniques have been evaluated by Heinrich et al. [34]. For large sets of multi-parameter points, Fua et al. [35] describe cluster-based hierarchical extensions of PCPs, which help to identify structure in large data more easily. Elmqvist and Fekete [36] discuss hierarchical aggregation approaches for information visualization, with applications to PCPs. More recently, Richer et al. [7] proposed a hierarchical aggregation framework, which enables the visual analysis of large datasets on distributed computer systems, and Sansen et al. [6] discuss the practical implementation of such systems on big data hardware.

To optimize rendering times, rather than visual display, Novotny and Hauser [3] and Blaas et al. [4] propose binning continuous data parameters and precomputation of 2D joint histograms for pairs of data parameters. PCPs are then rendered from the 2D joint histograms. Rubel et al. [37] propose an adaptive binning approach, and Cui et al. [38] use binning to create a web-based parallel-coordinates framework for visual data analysis. In this paper, we build upon binning and show that even for large datasets the joint histograms that are required for PCP rendering can be generated at interactive rates. Similar in spirit to previous work on the efficient computation of image histograms Scheuermann and Hensley [39], we employ parallelism and high memory bandwidth to scatter large sets of datapoints into GPU render targets. In contrast to Cui et al. [38], we demonstrate interactive PCP visualization while not relying on data abstractions, which might potentially discard important information in the data.

3. Method overview and data

The data is read from a collection of data files, in which the ensembles E_i are stored in succession as a sequence of their members. For each E_i, the parameter values of each member are read in the order of the datapoints in the input file. The parameters of each member are stored normalized in 1D half-precision floating-point arrays in CPU memory. Since these fields become large, they are internally partitioned into smaller blocks of 2^{26} entries each (Fig. 2, data pre-processing). For attributes that are defined on a subset of the dimensions of the dataset (i.e., a 2D field in a 3D dataset), the attribute domain is extruded and values are duplicated to retrieve attribute arrays of equal size.

Each block is uploaded to the GPU, where it is stored in a storage buffer. If not all blocks fit into the available GPU memory, they are stored on the CPU when sufficient CPU RAM is available. If this is not the case, the data is first compressed (Fig. 2, compression) using the publicly available CUDA compression library by Treib et al. [40], which we have re-implemented to run with the Vulkan API. It provides lossy GPU-based compression using a combination of the discrete wavelet transform, coefficient quantization, run-length encoding, and Huffman coding. GPU decompression is supported such that either the compressed data can be stored completely in main memory and streamed to the GPU to avoid expensive load operations from system drives or the data can be streamed efficiently from system drives to avoid the file I/O bottleneck. Section 5 shows the timings for differently sized datasets and different transfer operations. We have slightly modified the compression scheme to avoid splitting the blocks into smaller sub-blocks so that memory address indirections introduced by the use of a block table are avoided. We typically choose a compression ratio of roughly 1:10, resulting in a compression error below 0.5% of the value range of each parameter. We demonstrate in Section 5 the visual artifacts that can be introduced by a lossy compression scheme. Upon decoding a block, the data is stored in a storage buffer in local GPU memory and handled the same as uncompressed data by the following pipeline stages.

Assuming a set of K data parameters, plots must be rendered for $K-1$ pairs of adjacent coordinate axes. For each axis pair, multiple 2D render targets (i.e. storage buffers) are allocated and used to count the number of lines that are drawn between adjacent axes. Counting is realized on the GPU by letting a compute shader write (i.e., scatter) into the render targets using atomic increment counters (Fig. 2, scatter shader). We analyze in detail this approach and demonstrate that it can exploit GPU parallelism and high memory bandwidth to efficiently count the number of datapoints – within the currently selected parameter intervals – drawing the same line. In principle, the scattering operation can also be realized via point rendering using the GPU rasterizer. We briefly review and analyze this alternative, yet decided to not follow it due to its lower performance compared to compute shader-based scattering.

Eventually, a GPU line renderer (Fig. 2, line renderer) reads the information from the render targets and renders one line for every non-empty entry. The value in each entry is used to adjust a line's saturation or compute its opacity analytically. If the lines of different E_i should be distinguished in the final PCP, scattering and rendering is performed for these E_i consecutively, with their own user-defined color and opacity.

4. Scalable PCPs

The proposed PCP realization builds up on the observation that the number of visually distinguishable lines is determined by the pixel resolution available to display the coordinate axes, rather than the number of datapoints (see Fig. 3). Assuming that lines between adjacent parameter axes are rendered onto a screen region with resolution $T_u \times T_v$ pixels, where T_u represents the

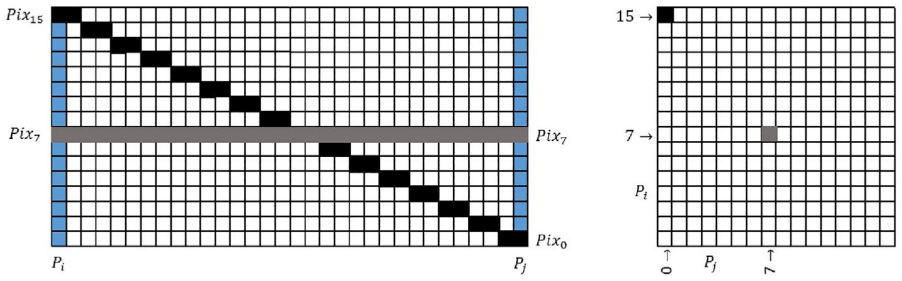

Fig. 3. Line representation. Left: Parallel coordinates axes for parameters P_i and P_j are shown in the pixel raster by blue pixels. Two lines (black and gray pixels) are drawn to represent two pairs of values in P_i and P_j, with the values mapped to pixels seven (15) on the axis of P_i and pixels 0 (seven) on the axis of P_j. Right: The two lines in the corresponding (P_i, P_j) raster.

height in pixels of the coordinate axes and T_v is the horizontal spacing in pixels between adjacent axes, then a pair of lines can hardly be distinguished visually if their end points on either axis fall into the same pixel. Thus, for large datasets many datapoints draw very similar lines, and the maximum number of distinguishable lines is roughly equal to T_u^2. This observation suggests that a viewport resolution-dependent discretization and binning of the datapoints into T_u bins along each coordinate axis can reduce the maximum number of lines to be rendered to T_u^2 at limited loss of visual quality. Even though rendering lines with end point positions on sub-pixel scale affect the rasterization marginally, we show in Section 5 that the visual difference between classical and binned PCPs is negligible. To let the lines' appearance, e.g., its opacity, reflect the number of datapoints drawing each line, this number has to be counted at run time. In the following, we describe how to efficiently realize this counting using 2D render targets (i.e., storage buffers on the GPU) of resolution $T_u \times T_u$, which accumulate in element (i, j) the number of datapoints drawing a line between pixel bins i and j of adjacent coordinate axes. To distinguish between different ensemble members, separate counters are generated for each of them. Then, the line sets can be rendered in member-specific colors, or in a user-set order that shows the lines of a selected member on top of all others.

4.1. Compute shader-based scattering

To efficiently count the number of datapoints drawing the same line, we utilize the massive data parallelism and I/O bandwidth on recent GPUs to simultaneously write many values to arbitrary target positions in multiple bound render targets. Recently, Schütz et al. [41] have proposed a compute shader-based approach for rendering large sets of 3D spatial points, demonstrating the extreme performance that can be achieved when rendering pixel-size point primitives. Inspired by the reported results, we have implemented a compute shader-based scattering operation on the GPU. Multiple so-called work groups are spawned, which are internally mapped to the GPUs compute units and can access work group-local memory in a fast way.

The compute shader reads all parameter values of its currently assigned datapoint from the parameter buffers and first evaluates the datapoint's activation, i.e., it exists if at least one parameter is not within the currently selected (brushed) parameter intervals. Then, for each of the $K - 1$ pairs of coordinate axes the vertical pixel position of the first and the second parameter are used as target addresses into the currently bound render target. Note that the compute shader can write to multiple render targets in one single pass, so that the parameter values need to be read only once and the activation is computed instantly in the shader. The operation for the buffer writeout is set to an atomic increment, which needs to be executed for every datapoint and render target this point is written to.

Due to the many atomic increment operations, we cannot achieve the scatter performance reported by Schütz et al. [41]. When rendering 3D point sets, the number of atomic operations can be reduced significantly by early depth-tests in the compute shader, i.e., by testing the depth value in the render target and existing when the render target already holds a lower value. In our application, the atomic operation needs to be performed for every value that is scattered into a render target.

4.2. Line renderer

On the GPU, we employ the functionality of the geometry shader stage to spawn a line from each input vertex. Per render target a draw call with as many vertices as entries is issued. From its position in the render target, every vertex can compute the position on both PCP axes for which the current render target holds the 2D histogram. The vertex shader – in the ith render pass – reads for each vertex the value of the corresponding entry from the ith render target and forwards the value to the geometry shader. Processing is stopped if this value is zero, otherwise, a second vertex is spawned and the positions of both vertices in the viewport to which the final PCP is rendered are computed (see Fig. 4). The geometry shader then issues a line primitive using the two transformed vertex coordinates.

When reading a line count of n, the accumulated opacity of n lines is first computed analytically in the shader program as

$$\alpha_n = 1 - (1 - \alpha_L)^n \quad (1)$$

with α_L being the opacity of a single line. Eq. (1) can be derived inductively via iterative application of multiplicative alpha blending. Finally, a single line is rendered with color $C_L \cdot \alpha_n$, where C_L is the selected line color for the current render target. Thus, at most T_u^2 lines are blended together in the framebuffer, which significantly reduces overdraw and framebuffer precision issues arising from this. As another option, the line renderer can consider the values stored in the render targets to control the saturation of each line, i.e., to let the saturation decrease with increasing line count.

4.3. Rasterization-based scattering

As an alternative to compute shader-based scattering, we have also implemented a rasterization-based approach. Similar in spirit to early approaches for computing histograms on the GPU [39], the implementation utilizes the point-rendering capabilities on the GPU to render many points in parallel at arbitrary target positions in the bound render targets. Therefore, we use a single vertex buffer that is stored in GPU memory and can be re-rendered in multiple rendering passes.

Initially, a vertex array is generated, which has the same size as the blocks into which the data has been partitioned (i.e., 2^{26}

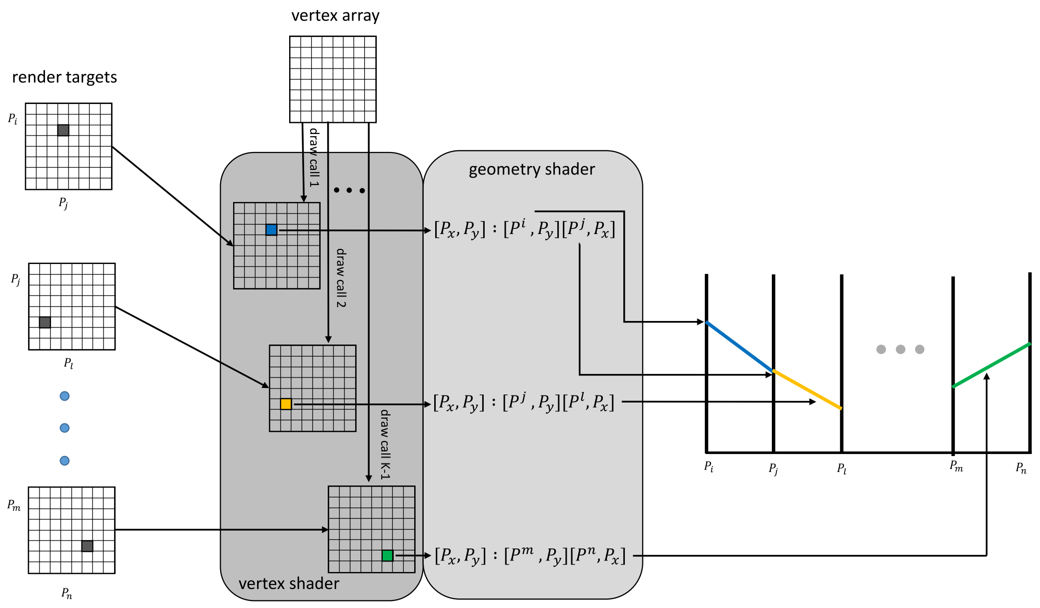

Fig. 4. GPU line renderer. A vertex buffer is rendered for every render target, and the geometry shader spawns a line for every non-empty entry. P^i to P^n are the viewport x-coordinates of the parameter axes.

entries). When using point rendering to realize the scattering operation, an additional bitfield – the datapoint activations – is required to indicate for every datapoint whether it is currently selected, i.e., whether all of its parameter values lie within the currently selected intervals. The bitfield is internally represented as an array of 32 bit units with one bit per datapoint and is reused for each block of datapoints to keep the memory requirements low. The reason for computing and storing the activations in each frame is that the pixel shader in the rasterization-based approach cannot scatter into multiple render targets at different positions simultaneously. Thus, scattering needs to be performed consecutively into the render targets, which would require to load all attribute values each time to evaluate the activation. By using the bitfield, the evaluation needs to be performed only once per datapoint, and the pixel shader needs to read only one single 32 bit element.

For a single pair of adjacent parameters in the PCP, the vertex array is rendered as many times as required to render one point primitive for every datapoint. The vertex shader first reads the point's activation and performs a bit-wise AND operation to obtain the value for the current vertex. Only if the bit is set, the shader proceeds by reading the parameter values of the currently rendered datapoint. The parameter values are then mapped to the discrete set of pixels that are used to represent the vertical parameter axes, and for the current pair of coordinate axes the pixel position of the first and the second parameter are used as target addresses into the currently bound render target. The state of the output merger is set so that an atomic addition operation is performed, and the output from the fragment shader equals 1 to emulate the atomic increment operation.

As reported by Schütz et al. [41] in the context of point cloud rendering, where every point represents an object sample in 3D space, rasterization-based point rendering can be accelerated significantly if the point set is laid out in a GPU friendly order. If the point set is laid out along a space-filling curve so that points close to each other in space are projected into pixels lying close together, distributed framebuffer writes and cache misses thereof can be reduced considerably. Furthermore, by splitting the sorted sequence into blocks and shuffling the processing order of these blocks, even lower rendering times are achieved due to a more uniform utilization of the GPUs viewport tile units.

In our application, however, these optimizations are problematic since the placement of a point within the render target depends on the parameter values. Following space-filling curves would thus be required in all visible 2D parameter sub-spaces in the PCP. Thus, multiple ordered sequences, and also one activation field per sequence, need to be stored, which significantly increases the memory requirement. Furthermore, if the axis ordering is changed, the whole layout for the affected parameter pairs has to be recomputed. Due to these reasons, we could not consider these otherwise effective improvements, resulting in about 10x lower performance of the rasterization-based approach compared to the compute shader-based approach.

4.4. Priority rendering

Priority polyline rendering [29,42] is an method to enhance the perception of proximity between datapoints in dense datasets. Upon selecting the value of a certain parameter, called the priority center, the datapoints are sorted with respect to decreasing distance of their value to the selected one. The polylines are then rendered in this order, so that lines belonging to datapoints with parameter values closer to the selection are rendered on top of those with a larger distance. The distance can be further mapped to color to emphasize those datapoints with a closer value.

In our approach, priority polyline rendering can be enabled with only a slight change in the scattering operation that is used to generate the render targets. Instead of an atomic increment operation, the write operation is performed with the distance to the selected parameter value as argument, and an atomic minimum operation is selected for the render targets. In this way, all render targets – regardless of which parameter pair they represent – always keep track of the minimum distance from the selected priority center. I.e., a value in a render target indicates that the corresponding line should be rendered, and states for all datapoints whose polyline contains this line what the minimum distance of any of these datapoints to the priority

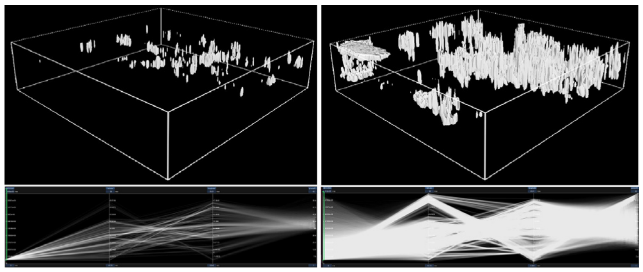

Fig. 5. Iso-surfaces to a value equal to 7.5×10^7 in NCCLOUD in two different members of the Grand Ensemble including a parallel coordinates visualization.

center is. For rendering the lines, the elements in the 2D render targets are sorted such that lines with a low distance are drawn last and appear in front of all other lines. Notably, this approach bypasses the need for exhaustive sorting operations on the full set of datapoints, since sorting is required only for the significantly smaller set of elements in the render targets.

While this approach does generate a correct priority plot for fully opaque lines, only information about the closest distances and no underlying distribution of the distances of all datapoints are stored and thus when reducing the opacity the resulting plot diverges from true priority rendering.

5. Results and evaluation

In the following, we first describe the dataset that was used to test the performance and scalability of our approach. We then provide a quantitative performance analysis of the stages of the proposed PCP realization, and continue with an illustration of the visual quality of the PCPs generated with the proposed methods, including the supplementary visualization options that have been discussed. All experiments have been carried out on a single node desktop PC with an Intel Xeon CPU at 3.60 GHz with 64 GB RAM, equipped with an Nvidia 3090 GPU with 24 GB on-chip memory.

5.1. Dataset

For the evaluation, we use a Grand Ensemble dataset that was produced by Matsunobu et al. [5] to study the impact of uncertainty in cloud microphysics models on convective weather forecasts. In their original study, the authors examine the effect of two microphysical model parameters on the uncertainty that is observed in ensemble simulations. The parameters are called CCN (density of cloud condensation nuclei) and NU (shape parameter of a Gamma distribution, describing the size distribution of cloud water droplets). For both input parameters three possible values are considered, and ensembles are generated for all possible pairs of input values. The dataset thus comprises a total of nine different simulation ensembles, with 20 members each. The difference between members within each sub-ensemble results from varying the boundary conditions (BC) used for the simulation. Each of the 180 ensemble members contains time-variate volumetric simulation output for 40 prognostic variables over a rectangular domain in central Europe, from which we select temperature, pressure, specific humidity, cloud density, total precipitation and 3D wind components for the analysis. The simulation data is stored at time steps of one hour on a (716×651)-grid with horizontal resolution of 2 km and 65 levels in height. In the current work we consider only one single time step including 33 horizontal levels from the ensemble, which amounts to 2.7 billion datapoints and 56 GB of memory. In addition to the seven prognostic variables, each datapoint is assigned additional descriptor variables, providing information about its horizontal (longitude and latitude) and vertical location (level), as well as the index of the ensemble member it belongs to. In Fig. 5, the 3D distribution of the particle density parameter NCCLOUD is illustrated via iso-surface renderings for two different ensemble members. Differences between the members, as well as a clustering of regions with high NCCLOUDS along the latitude dimension are clearly visible. To validate the accuracy of the proposed method for a dataset with smaller size, we use the cars dataset with seven attributes and 392 datapoints [43].

5.2. Performance evaluation

Reading the data from disk and subdividing each set of parameter fields into smaller blocks takes roughly 30 min. On the CPU, each block is optionally compressed using *cudaCompress*. This requires additional 15 min for the whole ensemble and reduces the memory requirement from 56 GB to 5.6 GB, so that even multiple time steps can be stored on the GPU. Streaming one compressed time step from the CPU to the GPU takes below one second, yet decoding the ensemble on the GPU, where a decompression throughput of approximately 5 GB/sec is achieved, requires about 10 s. Since uploading directly from RAM via PCI-e x16 4.0 yields a throughput of roughly 15 GB/sec on our hardware, streaming the uncompressed data is preferable in terms of speed and quality. However, when the data needs to be streamed from disk, typically not more than 1 GB/sec sustained bandwidth can be expected, even on recent SSD technologies. In such scenarios, compression becomes a useful option.

For different numbers of datapoints and vertical viewport resolutions, Fig. 6 shows the timings for scattering (i.e. generating the line counts in the render targets) and line rendering. All times are measured under the assumption that the uncompressed parameter fields are available in GPU memory.

As can be seen, the time required by the final line renderer varies only slightly and, in particular, is negligible compared to the time consumption of the scatter shader. Given a pixel resolution of $T_u \times T_v$ for the rendering of lines between two adjacent parameter axes, at most $T_u \times T_u$ lines need to be rendered for this parameter pair. Even when T_u equals 2^{10} and all lines are rendered, line rendering using x2 supersampling requires only about 100 milliseconds.

One can see that a full PCP update for the Grand Ensemble at a vertical resolution of up to 2000 pixels can be accomplished in slightly less than one second when all data can be stored in GPU memory. Beyond this number, interactivity degrades linearly with the number of datapoints. Since on our GPU target architecture, however, only approximately one billion datapoints can be stored uncompressed in GPU memory, in every frame we stream the entire dataset from the CPU to GPU memory. This requires about 3 s via PCI-e4 X16, so that an overall frame rate of roughly 4 s per frame is achieved.

When using priority rendering, once the scattering operation has been performed, the elements in the generated 2D render targets need to be sorted to draw the corresponding lines in the order of decreasing distance to the selected parameter value. In our current implementation, this is realized by downloading the render targets to the CPU, sorting the elements, and issuing a vertex buffer with the sorted sequence of lines to the GPU for rendering. The extra time that is required for downloading and sorting is about 380 milliseconds for an axis resolution of 1000 pixels and 11 attributes (i.e., ten 2D render targets). Notably, when priority rendering is selected, scattering is even faster than scattering using atomic increment operations, as early out tests regarding the current values in the render targets can significantly reduce the overall number of atomic operations that need to be performed.

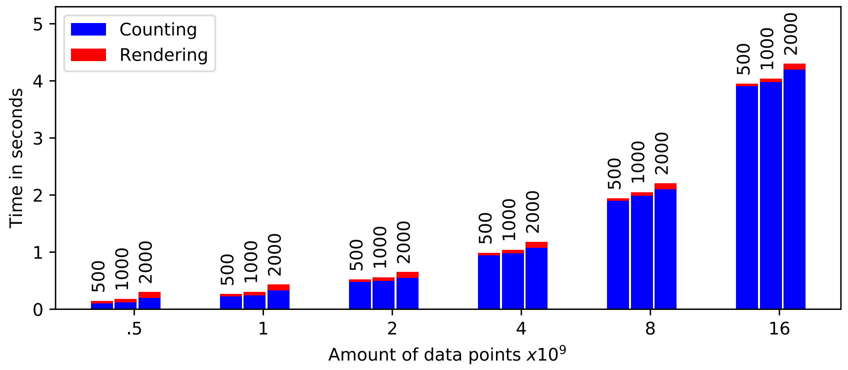

Fig. 6. Timing statistics for scalable PCPs, using different numbers of datapoints with 11 parameters each, and different resolutions of the viewport into which lines are rendered. Blue sub-bar: the time to scatter all datapoints into the render targets. Red sub-bar: the time to render all lines. It is assumed that all data is available uncompressed in GPU memory.

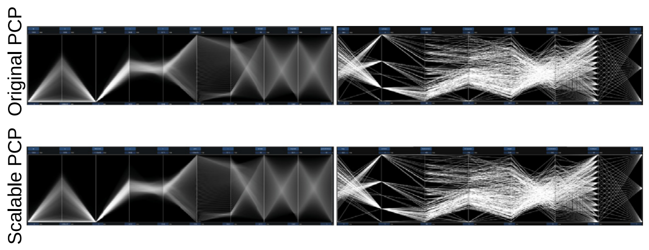

Fig. 7. Comparison of the standard PCP with one polyline rendered for each datapoint (top) and our proposed approach (bottom), using 30 million datapoints from the Grand Ensemble dataset (left) and a smaller dataset comprising 392 cars with seven attributes [43] (right). Line opacity is set to 10^{-5} (left) and 1 (right).

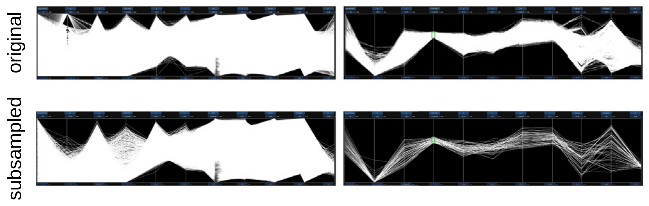

Fig. 8. Loss of outliers due to subsampling. Top: one ensemble of the Grand Ensemble dataset. Bottom: subsampling with a sampling rate of 1:100 is applied. Left: all datapoints are shown. Right: datapoints are thinned out via the selection of parameter sub-interval.

5.3. Qualitative analysis

First, we demonstrate that the proposed PCP realization produces the same results as the conventional approach where all polylines are rendered consecutively. In Fig. 7, datasets are rendered once with one polyline per datapoint, and once with the proposed binning approach. Both variants use the same line opacities. We use two different datasets with reduced number of datapoints to effectively reveal differences. While for the Grand Ensemble dataset the standard approach requires 32 bits per channel framebuffer precision to correctly blend the lines, 16 bits per channel are sufficient for our approach. Even for the smaller dataset with only NN datapoints, visual differences can hardly be perceived.

In Fig. 8, we further motivate our approach by demonstrating that even for large datasets subsampling can remove important information such as outliers. As can be seen, already a moderate subsampling rate of 1:100 causes loss of outlying datapoints. Furthermore, when the subsampled data is explored via parameter brushing, the selected sub-intervals can become so sparse that a meaningful visual analysis becomes impossible.

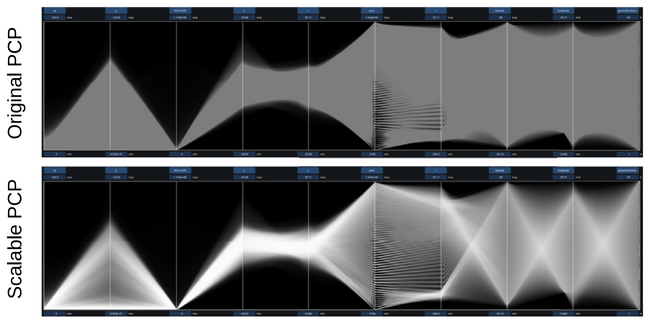

Fig. 9. Precision issues of standard PCP generation for 2.7 billion datapoints with a 16 bit framebuffer. Top: Standard PCP where one polyline is rendered for every datapoint. Bottom: Result of our new approach. Rendering times are 30 s (top) and one second (bottom).

For large numbers of datapoints, numerical inaccuracies due to insufficient framebuffer precision start to affect the quality of conventional PCPs (see Fig. 9). When using a 16 bit framebuffer and rendering all 2.7 billion lines of the Grand Ensemble, the numerical precision is quickly reached and information is lost, i.e., the PCP only gets darker when reducing the line opacity. Due to analytical alpha blending and a significantly reduced number of rendered lines, the scalable PCP realization shows no artifacts when reducing the opacity and can clearly reveal the major structures contained in the data. When rendering all 2.7 billion lines into a 32 bit framebuffer, visual artifacts disappear in the conventional PCP, but the rendering time is nearly tripled; it increases from 500 s to 1300 s for a 2000 pixels wide and 450 pixels high PCP, and rendering lines with one sample per pixel. In contrast, our approach renders the PCP in the same size and quality in less than one second.

Fig. 11 demonstrates the effect of lossy data compression on PCP accuracy, by comparing the quality of PCPs for different compression rates. Lossy compression is applied to each attribute separately, with compression ratio of roughly 5:1 and 10:1. The used compression scheme first transforms the original data into a wavelet representation to separate low-frequency and high-frequency signal components, and then performs a quantization of the resulting wavelet coefficients. To achieve higher compression rates, larger quantization steps are used, which directly affects the accuracy of the PCP. While a compression ratio of 5:1 shows no noticeable differences, the quantization of the wavelet

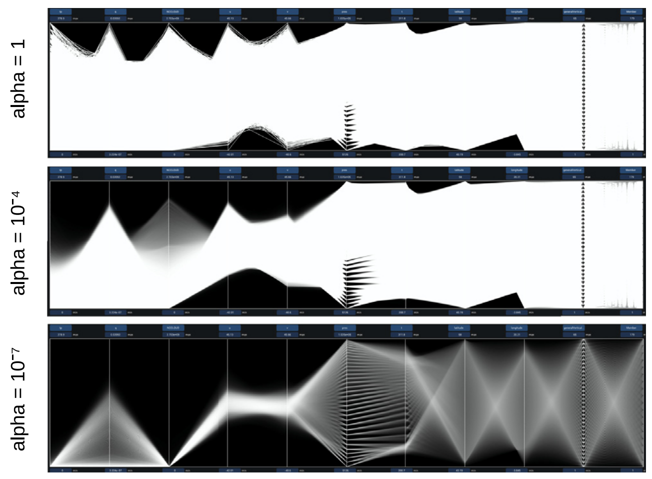

Fig. 10. Interactive opacity reduction reveals the major trends in certain multi-parameter combinations in the Grand Ensemble comprising 2.7 billion datapoints.

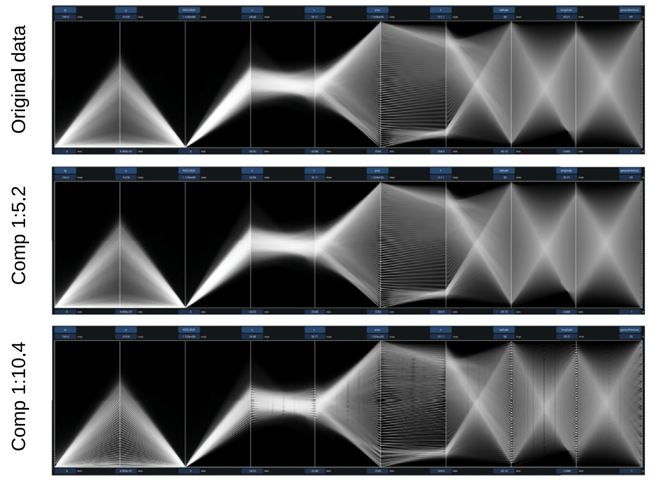

Fig. 11. Influence of compression rate on quality. From top to bottom: PCP using the original parameter values, and using the lossy compressed parameter values with a compression ratio of 5:1 and 10:1. All plots where created with a vertical pixel resolution of 1000 pixels. Due to compression errors, parameter values are discretized into false pixels along the parameter axes.

coefficients introduces clearly visible artifacts when the ratio is increased to 10:1. Even though the compression error on the data level is low, i.e., a fairly high PSNR of 39 dB with a maximum point-wise error below 5% of the value domains, line end points can fall into false pixels due to the quantization. Depending on the height in pixels of the parameter axes, the quantized pixel positions can be multiple pixels apart from the correct positions. On the other hand, since the major structures in the data are preserved, we see lossy compression as a useful option to quickly obtain a first overview of the data in scenarios where the original data cannot be stored on the CPU.

5.4. Use case

Fig. 10 illustrates how interactive opacity control in scalable PCPs can help to uncover the major trends and reveal local maxima in the data distribution of the Grand Ensemble. From top to bottom, all polylines of the Grand Ensemble are blended with decreasing opacity for each line. With lower opacity, only those lines drawn by many datapoints survive in the final plot, demonstrating both the frequency of occurrence of certain parameter

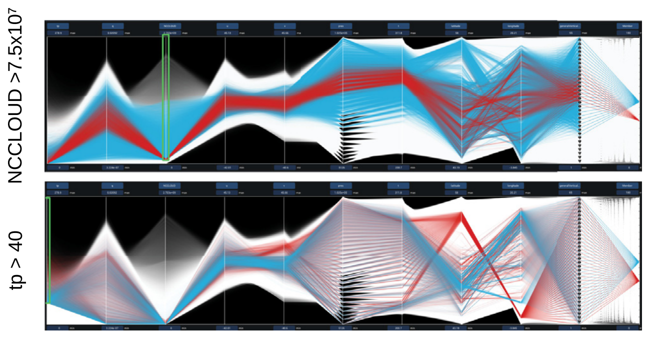

Fig. 12. PCP-based comparison of members in the Grand Ensemble. Top: Two members with specific NU, CCN and BC values are selected and rendered in blue and red, respectively. Datapoints where the number of cloud particles (NCCLOUD) is below 7.5×10^7 are discarded via brushing (green box). Bottom: the drawing order of the selected members has been changed, and brushing selects datapoints with total precipitation (tp) above 40. Datapoints in the members that are not selected are shown as background in white.

combinations and suggesting prominent correlation patterns between different parameters. An example for the latter is seen in the subplot pressure (pres) vs. temperature (t), where the majority of lines are parallel to one another, indicating a positive linear correlation between the parameters.

In the following, we demonstrate the use of the scalable PCP to analyze different members of the Grand Ensemble. Two exemplary ensemble members are selected by brushing along the right-most axis of the PCP. The datapoints of the selected members are rendered in red and blue, respectively, with the red lines on top of the blue lines. The same opacity value is used to allow for a better comparison of the number of datapoints in the appearing trends. Contextual information is provided by rendering lines for the complete dataset in white color as background. Note here, that the opacity value of both the background and the single members can be changed independently. Green boxes in the figure correspond to parameter brushes that can be applied either to the selected ensemble members or to the remaining context. This allows to quickly compare members regarding certain parameter ranges simultaneously.

In Fig. 12 (top), datapoints with high number of cloud particles (NCCLOUD, green box on 3rd axis) are compared between the selected members. Here one can instantly see, that the number of datapoints with such high values is larger in the member with a higher CCN value. Furthermore, the PCP shows that for one member there is a region at high latitude where high NCCLOUD values exist, whereas the other member does not contain datapoints in this region with the specific NCCLOUD values. The members are then compared with respect to total precipitation (tp), see Fig. 12 (middle). Since in the original data, precipitation is given only at a single horizontal layer, the values in this layer are copied across all other layers. In addition, the drawing order of the red and blue lines is switched, since the blue lines would have been vastly obscured otherwise. Again, a significant difference in the parameter domains can be seen. Looking at the spatial coordinate values a certain region at high latitudinal and low longitudinal values can be observed for the red member where precipitation above a value of 40 exists, while in the blue member this region did not include any high precipitation. Further the equal spread pattern between longitude and the parameter generalVerticalLayer hints at the copied values: each layer has the same tp field and thus, looking at the vertical layer, each layer has an equal amount of active points.

Finally, in Fig. 13 the entire Grand Ensemble is rendered with no particular members selected. Instead, brushing is conducted

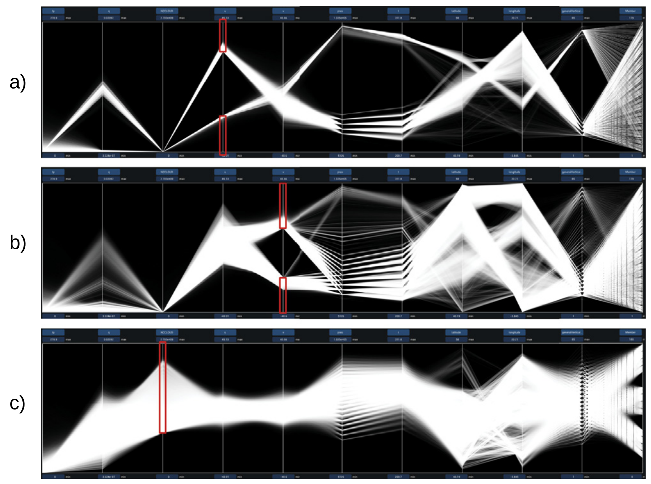

Fig. 13. Exploring the Grand Ensemble by interactively applying brushes to different attributes. In (a) u-wind extremes are selected, (b) shows v-wind extremes and in (c) high NCCLOUD values are selected.

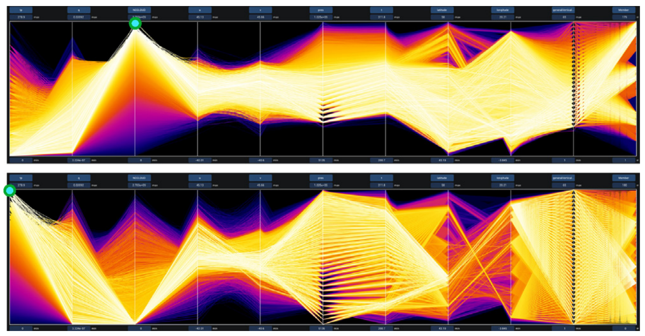

Fig. 14. PCPs of the Grand Ensemble including priority rendering. A value on a specific axis is selected as priority center (turquoise dots), and polylines are colored according to the distance of their value to the priority center, i.e., from low to high distance the color changes from white over red to blue. Polylines with lower distances are rendered on top of polylines with higher distances.

on different parameter axes. In (a) high absolute wind speeds in u direction were selected. Here one can see that these high wind speeds occur in all members (equal spread on the member axis) and are mainly apparent either in a very high or low vertical layer. When analyzing the dataset interactively and looking only at positive or negative wind speeds one can further see that in the low vertical layers only high positive wind speeds exist while all negative wind speeds occur in high layers. Also positive correlations between pressure (pres), temperature (t) and general vertical layer can be observed for high u wind speeds. In (b) high absolute wind speeds in v directions are shown. In comparison to (a), the spatial distributions of high positive and negative wind directions are not separated when looking at the vertical position in the 3D domain. Further no noticeable difference wrt. the simulation input parameters can be observed. In (c) then the correlation of NCCLOUD values is visualized. By choosing high NCCLOUD values and reducing the opacity value one can see a three-fold structure emerging on the simulation parameter axis (i.e., rightmost axis). Since along this axis the three simulation parameters CCN, NU and BC are laid out in BC-major order, it can be seen that high NCCLOUD values are given for three specific CCN/NU value combinations. Furthermore, the general trend of the multi-parameter dataset across all members for high NCCLOUD values is visualized by regions with high opacity. With this visualization also information about average temperature, average pressure and other parameters across multiple members in the dataset can be gained.

In Fig. 14, the Grand Ensemble is visualized using priority rendering in the PCP. The priority center is set to the maximum NCCLOUD value in the top plot and maximum tp value in the bottom plot. With priority rendering active, it is possible to track datapoints with respect to a priority center on a single axis which reveals correlations to attributes which are not direct neighbors of the priority center axis. Like this one can see that high NCCLOUD values correspond always to low total precipitation (tp) values. Further when looking at the member attribute a certain structure is apparent. This shows that high NCCLOUD values are directly influenced by certain combinations of simulation input parameters and are then consistent across the members of a single such combination. In the bottom plot one then can track datapoints with high tp values, revealing their correlation to low NCCLOUD values as well as showing certain latitudinal and longitudinal regions with high tp values. Further certain members with exceptional high tp values can be easily determined by the coloring at the member axis.

6. Conclusion and future work

We present a GPU optimized computation pipeline for creating multiple bi-dimensional histograms that are then used to generate parallel coordinates plots (PCPs) for large datasets at interactive framerates. Both a compute shader-based and rasterization-based approach are described and compared to each other regarding computational efficiency. By exploiting high memory bandwidth and compute parallelism on the GPU, PCPs for up to 3 billion datapoints with 11 parameters each can be generated in less than 1 s when all data fits into GPU memory. Due to algebraic blending of lines in a GPU shader instead of blending all lines in the output merger stage, framebuffer precision issues can be avoided. Thus, PCPs can be rendered into 16 bit framebuffers instead of 32 bit framebuffers, which gives a significant performance improvement.

We demonstrate the use of the pipeline for analyzing a Grand Ensemble comprising numerical weather forecast simulations, and show that the limiting factor is the memory available on the GPU and the CPU. Since not all data can be stored in GPU memory, the data needs to be streamed from the CPU via PCIe-4 x16. This reduces the throughput to roughly 0.7 billion datapoints per second, including streaming, histogram update and PCP rendering. To support the analysis of multiple time steps, which cannot even be stored in CPU-RAM, we provide the possibility to work on a compressed data representation that can be streamed at much higher rates from disk to the GPU than uncompressed possible.

In the future, we also aim to investigate the use of alternative lossy compression schemes like cuSZ [44] to improve the performance of the GPU decompression stage, at the same time enforcing upper bounds on the introduced compression errors. By this, the visual error can be reduced, and the overall runtime for datasets that need to be streamed from disk can be improved.

Another future direction will be to investigate other PCP visualizations such as continuous parallel coordinates by [45] and bundled PCPs, based on the fast bi-dimensional histogram calculation presented in this work. For continuous parallel coordinates the bi-dimensional histograms can be used as density estimators for the numerical integration of the dual line in the data domain. In this context, it will also be interesting to enable the integration of spline PCPs, which requires to consider more than 2 adjacent parameter axis in the scattering operation.

CRediT authorship contribution statement

Josef Stumpfegger: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – review & editing, Visualization. **Kevin Höhlein:** Conceptualization, Methodology, Formal analysis, Resource, Data curation, Writing. **George Craig:** Investigation, Supervision, Funding acquisition. **Rüdiger Westermann:** Conceptualization, Methodology, Writing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Funding by the DFG trans-regio project "Waves to Weather"

Data availability

The code will be published via a github link, an alternative synthetic dataset will be made available for testing.

Acknowledgments

This study was conducted within the subproject B5 of the Transregional Collaborative Research Center SFB/TRR 165 Waves to Weather funded by the German Research Foundation (DFG).

References

[1] Inselberg A, Dimsdale B. Parallel coordinates: A tool for visualizing multidimensional geometry. In: Proceedings of the first IEEE conference on visualization: Visualization90. IEEE; 1990, p. 361–78.
[2] Kumpf A, Stumpfegger J, Hartl PF, Westermann R. Visual analysis of multiparameter distributions across ensembles of 3D fields. IEEE Trans Vis Comput Graphics 2021.
[3] Novotny M, Hauser H. Outlier-preserving focus + context visualization in parallel coordinates. IEEE Trans Vis Comput Graphics 2006;12(5):893–900.
[4] Blaas J, Botha C, Post F. Extensions of parallel coordinates for interactive exploration of large multi-timepoint data sets. IEEE Trans Vis Comput Graphics 2008;14(6):1436–51.
[5] Matsunobu T, Zarboo A, Barthlott C, Keil C. Impact of combined microphysical uncertainties on convective clouds and precipitation in ICON-D2-EPS forecasts during different synoptic control. Weather Clim Dyn Discuss 2022;2022:1–25. http://dx.doi.org/10.5194/wcd-2022-17, URL: https://wcd.copernicus.org/preprints/wcd-2022-17/.
[6] Sansen J, Richer G, Jourde T, Lalanne F, Auber D, Bourqui R. Visual exploration of large multidimensional data using parallel coordinates on big data infrastructure. In: Informatics, vol. 4, (3):Multidisciplinary Digital Publishing Institute; 2017, p. 21.
[7] Richer G, Sansen J, Lalanne F, Auber D, Bourqui R. HiePaCo: Scalable hierarchical exploration in abstract parallel coordinates under budget constraints. Big Data Res 2019;17:1–17.
[8] Lachei and inmetak. Wavestoweather/PCViewer: V0.1-alpha. 2022, http://dx.doi.org/10.5281/zenodo.7225765.
[9] Liu S, Maljovec D, Wang B, Bremer P-T, Pascucci V. Visualizing high-dimensional data: Advances in the past decade. IEEE Trans Vis Comput Graphics 2016;23(3):1249–68.
[10] Obermaier H, Joy KI. Future challenges for ensemble visualization. IEEE Comput Graph Appl 2014;34(3):8–11.
[11] Wang J, Hazarika S, Li C, Shen H-W. Visualization and visual analysis of ensemble data: A survey. IEEE Trans Vis Comput Graphics 2018;25(9):2853–72.
[12] Wang J, Liu X, Shen H-W, Lin G. Multi-resolution climate ensemble parameter analysis with nested parallel coordinates plots. IEEE Trans Vis Comput Graphics 2016;23(1):81–90.
[13] Kumpf A, Stumpfegger J, Westermann R. Cluster-based analysis of multi-parameter distributions in cloud simulation ensembles. 2019.
[14] Inselberg A. The plane with parallel coordinates. Vis Comput 1985;1(2):69–91.
[15] Siirtola H, Räihä K-J. Interacting with parallel coordinates. Interact Comput 2006;18(6):1278–309.
[16] Elmqvist N, Stasko J, Tsigas P. DataMeadow: A visual canvas for analysis of large-scale multivariate data. Inf Vis 2008;7(1):18–33.
[17] Kandogan E. Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions. In: Proceedings of the IEEE information visualization symposium, vol. 650, Citeseer; 2000, p. 22.
[18] Wegman EJ. Hyperdimensional data analysis using parallel coordinates. J Amer Statist Assoc 1990;85(411):664–75.
[19] Yang J, Peng W, Ward MO, Rundensteiner EA. Interactive hierarchical dimension ordering, spacing and filtering for exploration of high dimensional datasets. In: IEEE symposium on information visualization 2003 (IEEE Cat. No. 03TH8714). IEEE; 2003, p. 105–12.
[20] Ferdosi BJ, Roerdink JB. Visualizing high-dimensional structures by dimension ordering and filtering using subspace analysis. In: Computer graphics forum. vol. 30, (3):Wiley Online Library; 2011, p. 1121–30.
[21] Long TV, Linsen L. Efficient reordering of parallel coordinates and its application to multidimensional biological data visualization. In: Visualization in medicine and life sciences III. Springer; 2016, p. 309–28.
[22] Lu LF, Huang ML, Zhang J. Two axes re-ordering methods in parallel coordinates plots. J Vis Lang Comput 2016;33:3–12.
[23] Theisel H. Higher order parallel coordinates. In: VMV. 2000, p. 415–20.
[24] Graham M, Kennedy J. Using curves to enhance parallel coordinate visualisations. In: Proceedings on seventh international conference on information visualization, 2003. IV 2003. IEEE; 2003, p. 10–6.
[25] McDonnell KT, Mueller K. Illustrative parallel coordinates. In: Computer graphics forum, vol. 27, (3):Wiley Online Library; 2008, p. 1031–8.
[26] Zhou H, Yuan X, Qu H, Cui W, Chen B. Visual clustering in parallel coordinates. In: Computer graphics forum, vol. 27, (3):Wiley Online Library; 2008, p. 1047–54.
[27] Luo Y, Weiskopf D, Zhang H, Kirkpatrick AE. Cluster visualization in parallel coordinates using curve bundles. IEEE Trans Vis Comput Graphics 2008;18.
[28] Yuan X, Guo P, Xiao H, Zhou H, Qu H. Scattering points in parallel coordinates. IEEE Trans Vis Comput Graphics 2009;15(6):1001–8.
[29] Roberts RC, Laramee RS, Smith GA, Brookes P, D'Cruze T. Smart brushing for parallel coordinates. IEEE Trans Vis Comput Graphics 2018;25(3):1575–90.
[30] Pomerenke D, Dennig FL, Keim DA, Fuchs J, Blumenschein M. Slope-dependent rendering of parallel coordinates to reduce density distortion and ghost clusters. In: 2019 IEEE visualization conference. VIS, IEEE; 2019, p. 86–90.
[31] Heinrich J, Weiskopf D. State of the art of parallel coordinates. In: Eurographics (state of the art reports). 2013, p. 95–116.
[32] Moustafa RE. Parallel coordinate and parallel coordinate density plots. Wiley Interdiscip Rev Comput Stat 2011;3(2):134–48.
[33] Palmas G, Bachynskyi M, Oulasvirta A, Seidel HP, Weinkauf T. An edge-bundling layout for interactive parallel coordinates. In: 2014 IEEE pacific visualization symposium. IEEE; 2014, p. 57–64.
[34] Heinrich J, Luo Y, Kirkpatrick AE, Zhang H, Weiskopf D. Evaluation of a bundling technique for parallel coordinates. 2011, arXiv preprint arXiv:1109.6073.
[35] Fua Y-H, Ward MO, Rundensteiner EA. Hierarchical parallel coordinates for exploration of large datasets. IEEE; 1999.
[36] Elmqvist N, Fekete J-D. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. IEEE Trans Vis Comput Graphics 2009;16(3):439–54.
[37] Rubel O, Wu K, Childs H, Meredith J, Geddes CG, Cormier-Michel E, et al. High performance multivariate visual data exploration for extremely large data. In: SC'08: Proceedings of the 2008 ACM/IEEE conference on supercomputing. IEEE; 2008, p. 1–12.
[38] Cui W, Strazdins G, Wang H. Confluent-drawing parallel coordinates: Web-based interactive visual analytics of large multi-dimensional data. 2019, arXiv preprint arXiv:1906.10017.
[39] Scheuermann T, Hensley J. Efficient histogram generation using scattering on GPUs. In: Proceedings of the 2007 symposium on interactive 3D graphics and games. 2007, p. 33–7.
[40] Treib M, Reichl F, Auer S, Westermann R. Interactive editing of GigaSample terrain fields. Comput Graph Forum 2012;31(2):383–92. http://dx.doi.org/10.1111/j.1467-8659.2012.03017.x, URL: http://diglib.eg.org/EG/CGF/volume31/issue2/v31i2pp383-392.pdf.
[41] Schütz M, Kerbl B, Wimmer M. Rendering point clouds with compute shaders and vertex order optimization. In: Computer graphics forum, vol. 40, (4):Wiley Online Library; 2021, p. 115–26.
[42] Hauser H, Ledermann F, Doleisch H. Angular brushing of extended parallel coordinates. In: IEEE symposium on information visualization, 2002. INFOVIS 2002. IEEE; 2002, p. 127–30.
[43] Xmdvtool homepage. URL: https://davis.wpi.edu/xmdv/datasets/cars.html.
[44] Tian J, Di S, Yu X, Rivera C, Zhao K, Jin S, et al. Optimizing error-bounded lossy compression for scientific data on GPUs. In: 2021 IEEE international conference on cluster computing. CLUSTER, Los Alamitos, CA, USA: IEEE Computer Society; 2021, p. 283–93. http://dx.doi.org/10.1109/Cluster48925.2021.00047, URL: https://doi.ieeecomputersociety.org/10.1109/Cluster48925.2021.00047.
[45] Heinrich J, Weiskopf D. Continuous parallel coordinates. IEEE Trans Vis Comput Graphics 2009;15(6):1531–8.