Visualizing Runtime Evolution Paths in a Multidimensional Space (Work In Progress Paper)

Hagen Tarner
University of Duisburg-Essen
Essen, Germany
hagen.tarner@paluno.de

Fabian Beck
University of Bamberg
Bamberg, Germany
fabian.beck@uni-bamberg.de

ABSTRACT

Runtime data of software systems is often of multivariate nature, describing different aspects of performance among other characteristics, and evolves along different versions or changes depending on the execution context. This poses a challenge for visualizations, which are typically only two- or three-dimensional. Using dimensionality reduction, we project the multivariate runtime data to 2D and visualize the result in a scatter plot. To show changes over time, we apply the projection to multiple timestamps and connect temporally adjacent points to form trajectories. This allows for cluster and outlier detection, analysis of co-evolution, and finding temporal patterns. While projected temporal trajectories have been applied to other domains before, we use it to visualize software evolution and execution context changes as *evolution paths*. We experiment with and report results of two application examples: (I) the runtime evolution along different versions of components from the *Apache Commons* project, and (II) a benchmark suite from scientific visualization comparing different rendering techniques along camera paths.

CCS CONCEPTS

• **Human-centered computing** → **Information visualization**;
• **Software and its engineering** → *Software performance.*

KEYWORDS

Dimensionality reduction, software visualization, runtime performance, dynamic analysis, software evolution

1 INTRODUCTION

Runtime characteristics and performance of a software system can be described by different metrics: time it took to execute a piece of code, invocation counts, memory and energy consumption, idle and waiting times, and many more. Usually, it is not enough to represent the behavior of the system along one metric only, but multiple ones need to be shown to provide a rich picture. Visualizing many metrics at once gets more challenging with the number of recorded metrics and finer granularity of recording (e.g., system vs. method level). A viable solution can be to transform the data first and project it from a space with an arbitrary number of dimensions (one per recorded metric) to a visualizable two- or three-dimensional space (Figure 1, left). Though simplifying, this can preserve similarities and outliers regarding the behavior of visualized software entities.

When considering, however, that the dynamic behavior of a software also alters with every change of code and execution in a different setup, the problem gets more challenging again. Connecting the metrics along a temporal dimension, this yields a multivariate time series, which can be described as an *evolution path* through the multidimensional space. After projection, it can be visualized as a trajectory within the visualized sets of data points (Figure 1, right). Different types of trajectory will allow identifying, for example, substantial changes in behavior, clusters of co-evolution, trends of improvement and regression, as well as stable behavior.

In this paper, we explore the potential of projection-based multivariate time series visualizations for understanding evolutionary changes in runtime behavior. Using projected *evolution paths*, it captures the variety of runtime statistics through dimensionality reduction (DR) and visualizes changing behavior as lines connecting the points of a scatter plot. While similar visualization approaches have been used in other domains already [5], we are not aware of any application to software engineering data. We combine this approach with different color mappings and complement it with parallel coordinates plots (PCP), which help us understand details of a selection of points. Visual patterns appear—similar to those described by previous works [2, 5]—and guide the identification of relevant insights. We demonstrate the usefulness of the approach with results from two applications, the evolving behavior of method-level code entities in a software project and the changing performance of rendering techniques along camera paths. While these results are promising already, we only see the work as a first step towards a full-fledged visual analytics solution.

The prototype is implemented as a web-based tool and available online: https://vis-uni-bamberg.github.io/evolution-paths/

2 RELATED WORK

Dimensionality reduction is a standard technique to project multivariate data to two dimensions for visualization in a scatter plot. If applied to time-dependent multivariate data, connecting the temporally adjacent points in the projection yields imaginary trajectories, often called *paths*. TimeCluster [1] uses a combination of deep learning and dimensionality reduction to visualize multivariate time series from biology and medicine as paths. TimeSeriesPaths [3] uses Principal Component Analysis (PCA) as the main reduction technique to project long and periodic time series for visual cluster analysis. Time Curves [2] finds patterns in the path of a single time series, that are projected using Multidimensional Scaling (MDS). Recently, ProjectionPathExplorer [5] uses t-Distributed Stochastic Neighbor Embedding (t-SNE) as well as Uniform Manifold Approximation and Projection (UMAP) [9] in an interactive visual analytics application for path exploration. It is used to analyze datasets from domains such as puzzles, games, and artificial intelligence. In the context of performance engineering, dimensionality reduction (namely PCA), has already been used to identify outliers in load testing data and aid analysts in comprehension [7, 8].

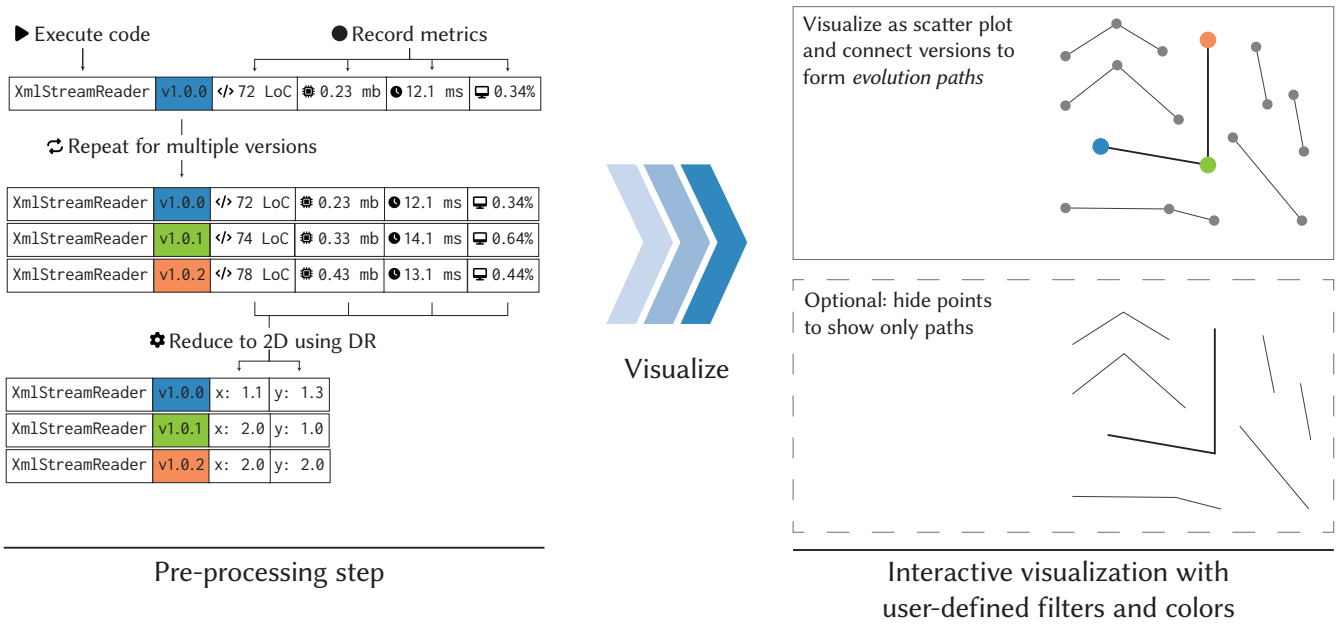

Figure 1: We consider the evolution of high-dimensional performance data of software artifacts and reduce the multidimensional vectors containing the recorded metrics down to two dimensions. In the resulting scatter plot, we connect adjacent software versions to form temporal trajectories, that we call *evolution paths*.

Our work builds on these methods for visualizing paths in multidimensional data spaces and adapts them for visualizing evolving runtime behavior. In a similar software visualization context, UMAP has also been used to project multivariate performance data [15]. This approach, however, does not consider the temporal dimension during visualization of the projection result. Focusing only on a small and specific set of performance and change metrics, evolving performance changes can be presented in detail as a matrix [12] or, aggregated on a high level, enriching commit messages [13]. Other approaches show evolving multivariate software metrics in a context of image rendering techniques [14], or limited to a coarser level of granularity and no behavior metrics [6, 11].

3 VISUALIZING EVOLUTION PATHS

In our approach, we collect evolving multivariate metrics on a potentially fine level of granularity (e.g., for many software artifacts or settings) and transform them through dimensionality reduction. The results are visualized as a scatter plot, in which we connect consecutive points as paths (see Figure 2). We implemented a prototype of this approach as a web-based application using *D3.js*[1] and *Vue.js*[2]. Besides the main visualization, the prototype also contains different interactive controls for data-driven filtering of the data and offers the user the ability to select multiple evolution paths by brushing via mouse. Usage of color throughout the whole application is user-defined and can be adapted to further the analysis process (see Section 3.2). For the dimensionality reduction performed in a pre-processing step, we use the Python implementation of UMAP and AlignedUMAP from the *umap-learn*[3] package. The projected datasets and the source code of the web-application are available on GitHub: https://github.com/vis-uni-bamberg/evolution-paths

The approach is open to diverse sets of multivariate data that somehow describe the behavior of a software and change over time. These can be mixtures of performance metrics and other dynamic or static metrics. Moreover, metrics that characterize the evolution of the software can be relevant to add, too. Since all metrics get fused, they should provide characteristic profiles that matter for a certain analysis goal. For instance, mixing performance metrics with metrics of code complexity can reveal clusters of similarly time-consuming and complex routines, and whether these co-evolve consistently. Likewise, the level of granularity of recording depends on the specific analysis and use case—evolution can be described regarding individual methods of a system or whole systems. Our approach generally targets to scale to at least a few hundred evolving software entities, each consisting of up to a few dozens of evolution steps (timestamps).

3.1 Data Model and Transformation

Our data model builds upon previously established frameworks for time series visualization and dimensionality-reduced time series [2, 5, 10]. Specifically, we use the notation proposed by Hinterreiter et al. [5]. A single *evolution path* $P = p_1, \ldots, p_n$ is an ordered set of time points. Evolution paths can be of differing lengths, e.g., when not all timestamps are present. Each time point $p_i = (t_i, s_i)$ consists of one timestamp $t_i \in \mathbb{R}$ and one multivariate data point

[1] https://d3js.org
[2] https://vuejs.org
[3] https://umap-learn.readthedocs.io/

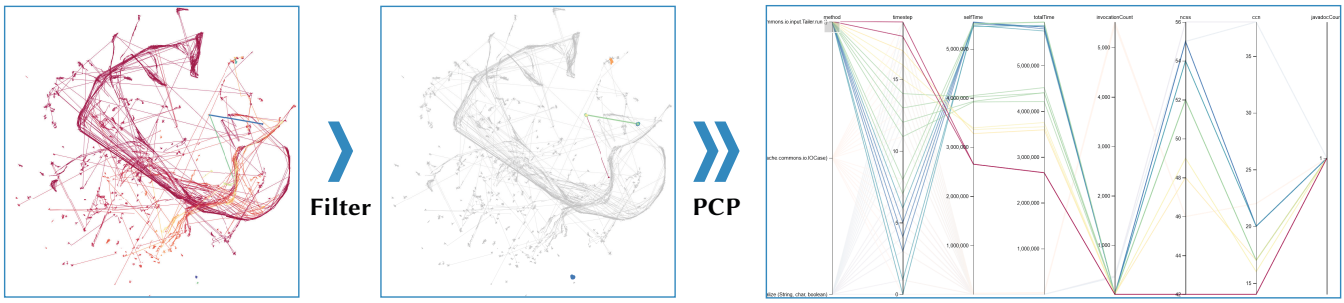

Figure 2: Exemplary workflow to analyze evolution paths: after metric-based coloring and scaling of the paths, data-driven filtering focuses the analysis on a few paths; the raw data of the selection can be inspected in the parallel coordinates plot of the details-on-demand view.

$s_i \in \mathbb{S}$. All data points have the same number of dimensions and contain the same set of recorded metrics. To avoid outliers distorting the distance calculation, we normalize the variables using the `RobustScaler` from the *scikit-learn*[4] package. Through experiments, we found Euclidean distance to be a good candidate for the distance metric d for our application examples. The embedding function f depends on the application example: we use UMAP in its original variant, as well as in the AlignedUMAP variant. Initial experiments with other dimensionality reduction techniques (namely, PCA and t-SNE, inspired by [5]) did not show as promising results. Only UMAP was able to preserve the global structure and local detail of the high-dimensional data in the projection. For visualization purposes, we define the projection space to have exactly two dimensions and configure UMAP to output one two-dimensional vector for every time point p_i. These two dimensions define the x and y coordinates in the resulting scatter plot (see Figure 1).

Generally, UMAP is a popular state-of-the art technique for dimensionality reduction. The original technique has been extended in the AlignedUMAP variant to project high-dimensional data with a temporal dimension. AlignedUMAP projects multiple timestamps at once and applies constraints to the position of temporally adjacent points to not differ too much in the final projection. This makes AlignedUMAP seem like a suitable candidate to project multivariate data with a temporal aspect, there is one caveat however: the technique expects a relation mapping to identify points in all timestamps and does not project points that occur in only one or non-adjacent timestamps. Depending on the dataset, this affects usability of the resulting projection.

3.2 Visual Encodings and Interactions

The main visualization of our approach is a scatter plot that contains all time points of all paths. To visualize a single path P, we connect all projected consecutive time points p_i via straight lines. Each line segment between two temporally adjacent points can be individually colored (see Figure 6, where path segments are colored by timestamp). Users can also select a metric of the data point s_i for scaling the line thickness of a path. To account for outliers dominating the scales, we offer different types of scale transformations: linear, squared, cubic, or logarithmic.

[4] https://scikit-learn.org

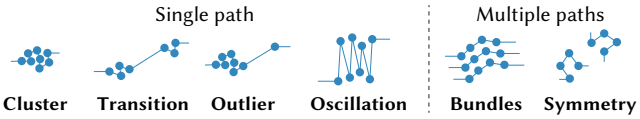

Figure 3: Visual patterns of single and multiple evolution paths.

To analyze parts of interest more closely, a user can sub-select paths by brushing or using the interactive metric-based filter components: selecting *top n* paths by a metric or giving upper and lower bounds for the range of each metric. Paths outside the current selection are grayed out, to reduce visual clutter but still give context. Selected paths can be further analyzed in a details-on-demand view that features a parallel coordinates plot and a data table, showing the raw data for each time point in the evolution path (see Figure 2 for an exemplary workflow).

3.3 Visual Patterns

Evolution paths in multivariate data might show characteristic and recognizable visual patterns (see Figure 3). Bach et al. [2] describe seven patterns for single paths, from which four show in our data and are relevant to our application (**Cluster**, **Transition**, **Outlier**, **Oscillation**). Hinterreiter et al. [5] extend this collection to include patterns from multiple paths (**Bundles**, **Symmetry**).

The **Cluster** pattern describes a path (or a subsequence of a path) where all points are located closely to each other in the projection. Some clusters are split by a gap, forming new clusters of multiple points on each side of it (**Transition**). A single point at the end of a transition is an **Outlier**. **Oscillation** creates a typical *zigzag* pattern along a path. Regarding multiple paths, **Bundles** occur when a group of paths shares visually similar segments in close proximity. In case some segments share visual similarity, except for translation and rotation, the **Symmetry** pattern can be observed.

4 RESULTS

We demonstrate our approach by describing exemplary exploration scenarios for two datasets from different software-related domains: software engineering and scientific visualization.

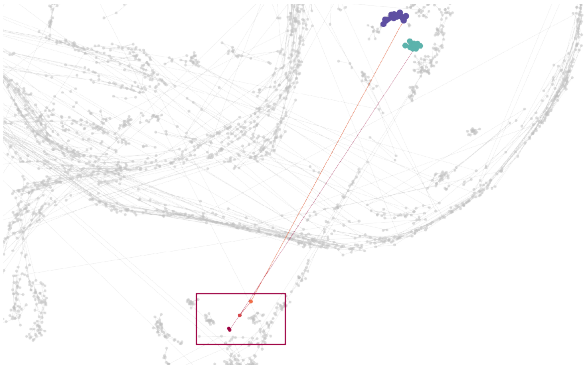

Figure 4: Outlier pattern observed in the *Apache Commons IO* dataset, marked by a red rectangle. Evolution paths are colored by invocation count.

4.1 Software Engineering: Apache Commons Projects

For the software engineering domain, we use the same dataset as Tarner et al. [15]. It contains data collected from three *Apache Commons* projects, from which we exemplarily analyze *Apache Commons IO*[5] in the following. The data contains a set of static and dynamic metrics on method level. Static metrics (number of non-commenting source statements, *Cyclomatic Complexity Number*, and count of *Javadoc* comments) were collected with *JavaNCSS*[6]. For measuring the dynamic metrics (self time, total time, and invocation count) all unit tests were executed and data was recorded via *VisualVM*[7]. Both sets of metrics were recorded for multiple commits per project. The data consists of 703 evolution paths (P) on method level recorded over 17 commits (t_i). This yields 10,567 time points (p_i) in total, each with six recorded metrics. We use UMAP[8] with a high number of neighbors for manifold construction and optimized settings for distance and spread of the projection to reduce overplotting (see Table 1).

Loading the dataset shows a projection that is dominated by the **Bundle** pattern (see Figure 2). We start our exploratory analysis of the data by using the *Non-Commenting Source Statements* (NCSS) metric to color the path segments, revealing that most evolution paths have a low number of NCSS (paths in red), and only a few paths show high numbers. To focus the analysis on the previously highlighted methods and reduce visual clutter, we filter the dataset to the top three methods by NCSS (second picture in Figure 2; the rest of the paths are now grayed out). Two of the three methods show the **Cluster** pattern. Further inspection of the PCP reveals that their evolution contains no significant changes, implying that these large methods have stayed mostly unaltered in code and behavior. The third method is different: the path shows two **Clusters** and one **Outlier**. All connected by two **Transitions**. The PCP of the raw data confirms the patterns. The method shows three distinguishable groups of timestamps for the NCSS metric. Early timestamps in the range of 52 NCSS to 56 NCSS (**Cluster**), timestamps from the middle of the dataset in range 48 NCSS to 49 NCSS (**Cluster**), and the last two timestamps both at the lowest value of 42 NCSS (**Outlier**), a reduction of 25% from the top value—developers might have refactored the code.

Outside the main **Bundle**, many isolated **Clusters** and **Transitions** or **Outliers** can be observed. The isolated **Clusters** are methods that do not change their characteristics in the observed timespan. The **Outliers** can be identified by singular long path segments. The frequency of these elongated segments hints at a lot of metric changes during a single evolution path's lifetime. A possible reason for this can be found when filtering the paths by commit: two consecutive commits are missing from the dataset. The evolution from t_8 to t_9 includes not one but three commits; possibly leading to more code changes being covered by a single timestamp.

Using a dynamic metric for coloring and scaling makes changes in behavior of methods visible (see Figure 4). Two methods from the `XmlStreamReader` class, for example, show anomalies for the last timestamp. Methods `getContentTypeEncoding` and `getContentTypeMime` both have high invocation counts for the first seventeen analyzed commits and drop to around 75% of that in the last two timestamps. This is visible in Figure 4, where both show a **Cluster** pattern with high invocation counts in the top and then transition to lower values in the bottom (marked by the red rectangle). This might relate to changed unit tests (selected methods being not called as often). This insight could help a developer evaluate whether the test coverage has changed and needs to be investigated.

Table 1: Hyperparameter settings used for UMAP.

Example	Variant	n_neighbors	min_dist	spread
SE	UMAP	1000[1]	1.0	5
SciVis	AlignedUMAP[2]	25	1.0	5

[1] 1000 equals to roughly a tenth of the dataset.
[2] The `alignment_window_size` parameter of AlignedUMAP was set to 3.

4.2 Scientific Visualization: TRRojan Benchmark Suite

Bruder et al. [4] published the *TRRojan* benchmark suite, which contains performance measurements of different techniques for particle rendering. The suite includes the systematic sampling of a set of performance-influencing factors for several datasets. Each univariate data point in the suite contains a *frames per second* (fps) value and is identified by its dataset, the rendering device, the camera path and sampling point along the path, the viewport resolution, and the used rendering technique.

For this application example, we focus on contrasting the used rendering technique. Evaluating the runtime performance of different techniques is essential when choosing a suitable technique for a given scenario or evaluating novel techniques. While the set of rendering techniques provides multivariate runtime data, the temporal dimension is provided through the camera paths. They

[5] https://commons.apache.org/proper/commons-io/
[6] http://www.kclee.de/clemens/java/javancss/
[7] https://visualvm.github.io/
[8] The dataset contains methods that are not available in all timestamps. Hence, as discussed in Section 3, we use UMAP for dimensionality reduction.

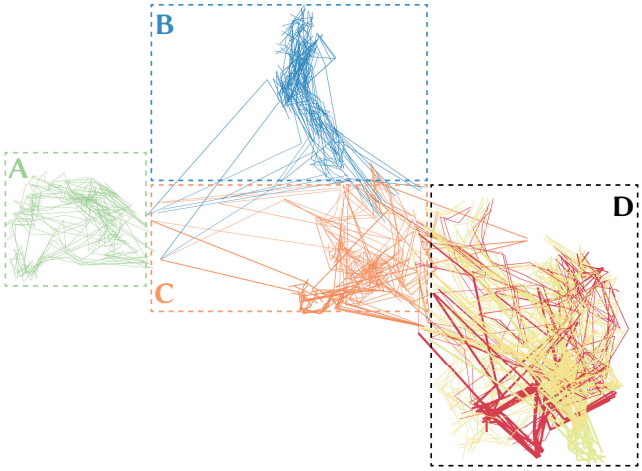

Figure 5: *Droplet* dataset, colored by *device*. Clusters have been marked by borders for reference.

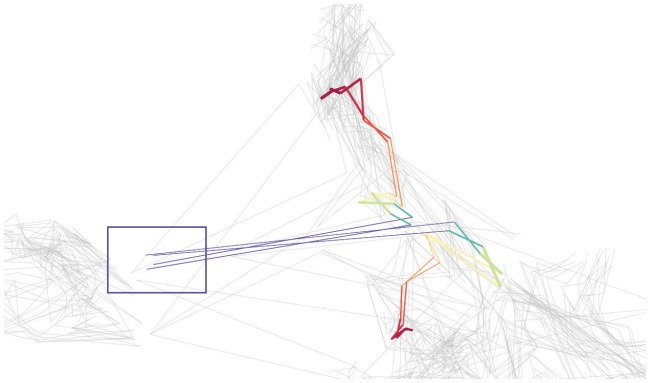

Figure 6: The *Droplet* dataset, colored by timestamp. Selected evolution paths show symmetric pairs, and a bundle of segments leading to an outlier (marked by purple rectangle).

place the performance measurements into a logical context and sequence of realistic usage. As an example for analysis, we use the *Droplet* dataset from the *TRRojan* suite. This dataset contains roughly 80k particles, which form three droplets of a liquid. The particles are rendered as spherical glyphs. To construct a multivariate vector with one dimension per rendering as a data point s_i from the univariate data, we aggregate the data points across devices and resolutions at each sampled camera position of each camera path. Temporal order t_i of the time points $p_i = (t_i, s_i)$ is given by the movement of the camera through the scene. In addition to the five compared rendering techniques, the data contains eleven camera paths (each sampled at eleven positions), six rendering devices, and three viewport resolutions. This yields 198 evolution paths in total. For projection, we use AlignedUMAP (see Table 1 for details on the used parameters). We consider the last three and next three timestamps for embedding with the AlignedUMAP variant.

The initial visualization of the dataset shows two clusters and a sparsely populated area between them. To start the exploration, we experiment with different metrics to drive the coloring of the plot. We find that coloring by rendering device further separates the clusters into four distinct groups (see Figure 5). All evolution paths in cluster A were rendered using a *Radeon (TM) RX 480 Graphics* GPU, cluster **B** only uses a *Radeon Vega Frontier Edition* GPU, cluster C *NVIDIA Quadro M6000 24GB* GPU, and cluster **D** contains all other GPUs (*NVIDIA TITAN Xp*, *NVIDIA TITAN X (Pascal)*, and *NVIDIA GeForce GTX 1080 Ti*). Further investigation of the area between clusters A, **B**, and C reveals that it is dominated by the **Transition** and **Outlier** patterns. Closer examination of the evolution paths in the details-on-demand view confirms that the time points in that region exhibit unusual behavior for at least one of the five rendering techniques. In this selection of evolution paths, four paths stand out because of their geometric characteristics (see Figure 6). All four of them converge in their last timestamp at a similar region (highlighted in purple rectangle) in projected space (**Bundle**), while showing different behavior for prior timestamps. The four paths can be divided into two pairs of two, based on path characteristics in earlier timestamps (**Symmetry**). While not being completely symmetric, the two paths of each pair share general similarity in these timestamps. The details-on-demand confirms: the two pairs differ in rendering device used (one is of cluster B, one of cluster C in Figure 5) and camera path (*diagonal y* and *diagonal z*), while all of them were rendered in 2048 × 2048 px resolution. Last timestamps show a significant drop in fps for all rendering techniques, with the *Screen-aligned quad* technique showing the largest drop from its usual range of 1000 fps to 5000 fps down to 6 fps.

This insight can be a starting point for the developer of the techniques for further investigation into why all rendering techniques perform poorly for the last camera configuration of the *diagonal y* and *diagonal z* camera paths, with a focus on the *Screen-aligned quad* technique. A possible explanation for the difference in performance is that, in the last steps of these paths, the camera gets close to the spherical glyphs of the rendered particles. Looking at the rendered result, we can confirm that the *Screen-aligned quad* technique shows different clipping behavior, than the rest of the techniques, resulting in more glyphs being drawn on screen. Further investigating the source code of this technique, it becomes apparent that it generates sprites for the back faces of the glyphs, increasing rendering time by creating unnecessary draw calls.

5 DISCUSSION AND FUTURE WORK

In this paper, we presented first steps towards visualizing the evolution of a software system's multivariate runtime characteristics using dimensionality reduction techniques. By connecting the projected points, they form evolution paths. To showcase applicability, we demonstrated the idea in the context of software engineering and scientific visualization. With this, we have, however, only verified the potential and general applicability of the idea, but not yet investigated any of its applications in detail. Specifically, it would be necessary to connect the visual patterns we observed to actionable insights for software engineers. Specific use cases (e.g., identifying performance regressions) likely require tailored settings and encodings.

Our approach already includes basic brushing-and-linking functionalities in combination with a parallel coordinates plot for a

details-on-demand view. Furthermore, data-driven filtering capabilities have also been added. However, for more efficiently performing analyses at a deeper level, we suggest extending our approach into a full-fledged visual analytics system. This proposed system could use custom tailored visualizations for the specific application (e.g., source code view and code diffs for code-related analyses) and advanced interactions (e.g., interactive bundle selection). Visual summaries of selected paths or groups of paths would further support reasoning, and enabling automatic similarity search for evolution paths could help in finding co-evolving software entities in the dataset. Adding these extensions to the proposed system, would allow for deeper analysis inside the system without the need to leave it (which was still needed to explain the findings in Section 4.2). A limitation of the current approach is that dimensionality reduction is done in a pre-processing step and thus not modifiable during an analysis session.

Another promising direction is applying the approach to other software-related scenarios. For example, when analyzing multivariate performance data of complex systems under different loads, the evolution paths can uncover unexpected anomalies. Also, it could help observe if state changes in modern single-page web application frameworks reflect alike in changes in the execution behavior of the frontend.

ACKNOWLEDGMENTS

We would like to thank André van Hoorn for providing feedback on an early version of the manuscript. This work is partly funded by MERCUR (project: "Vergleichende Analyse dynamischer Netzwerkstrukturen im Zusammenspiel statistischer und visueller Methoden") and Deutsche Forschungsgemeinschaft (DFG) as part of research grant 288909335.

REFERENCES

[1] Mohammed Ali, Mark W. Jones, Xianghua Xie, and Mark Williams. 2019. Timecluster: Dimension Reduction Applied to Temporal Data for Visual Analytics. *The Visual Computer* 35, 6-8 (2019), 1013–1026. https://doi.org/10.1007/s00371-019-01673-y

[2] Benjamin Bach, Conglei Shi, Nicolas Heulot, Tara Madhyastha, Tom Grabowski, and Pierre Dragicevic. 2016. Time Curves: Folding Time to Visualize Patterns of Temporal Evolution in Data. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (Jan. 2016), 559–568. https://doi.org/10.1109/TVCG.2015.2467851

[3] Jürgen Bernard, Nils Wilhelm, Maximilian Scherer, Thorsten May, and Tobias Schreck. 2012. TimeSeriesPaths: Projection-Based Explorative Analysis of Multivariate Time Series Data. In *Journal of WSCG (JWSCG)*. 97–106.

[4] Valentin Bruder, Christoph Müller, Steffen Frey, and Thomas Ertl. 2019. On Evaluating Runtime Performance of Interactive Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 26 (2019), 2848–2862. Issue 9. https://doi.org/10.1109/TVCG.2019.2898435

[5] Andreas Hinterreiter, Christian Steinparz, Moritz Schöfl, Holger Stitz, and Marc Streit. 2021. Projection Path Explorer: Exploring Visual Patterns in Projected Decision-Making Paths. *ACM Trans. Interact. Intell. Syst.* 11, 3–4, Article 22 (sep 2021), 29 pages. https://doi.org/10.1145/3387165

[6] Michele Lanza. 2001. The Evolution Matrix: Recovering Software Evolution Using Software Visualization Techniques. In *Proceedings of the 4th International Workshop on Principles of Software Evolution (IWPSE)*. ACM, Vienna, Austria, 37–42. https://doi.org/10.1145/602461.602467

[7] Haroon Malik, Hadi Hemmati, and Ahmed E. Hassan. 2013. Automatic Detection of Performance Deviations in the Load Testing of Large Scale Systems. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 1012–1021. https://doi.org/10.1109/ICSE.2013.6606651

[8] Haroon Malik, Zhen Ming Jiang, Bram Adams, Ahmed E. Hassan, Parminder Flora, and Gilbert Hamann. 2010. Automatic Comparison of Load Tests to Support the Performance Analysis of Large Enterprise Systems. In *14th European Conference on Software Maintenance and Reengineering (CSMR 2010)*. IEEE, Madrid, 222–231. https://doi.org/10.1109/CSMR.2010.39

[9] Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. https://doi.org/10.48550/ARXIV.1802.03426

[10] W. Müller and H. Schumann. 2003. Visualization Methods for Time-Dependent Data - an Overview. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*. IEEE, New Orleans, LA, USA, 737–745. https://doi.org/10.1109/WSC.2003.1261490

[11] Martin Pinzger, Harald Gall, Michael Fischer, and Michele Lanza. 2005. Visualizing Multiple Evolution Metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization (SoftVis)*. ACM, Saint Louis, Missouri, USA, 67–75. https://doi.org/10.1145/1056018.1056027

[12] Juan Pablo Sandoval Alcocer, Fabian Beck, and Alexandre Bergel. 2019. Performance Evolution Matrix: Visualizing Performance Variations along Software Versions. In *Proceedings of the 2019 Working Conference on Software Visualization (VISSOFT)*. IEEE, Cleveland, USA, 1–11. https://doi.org/10.1109/VISSOFT.2019.00009

[13] Juan Pablo Sandoval Alcocer, Harold Jaimes Camacho, Diego Costa, Alexandre Bergel, and Fabian Beck. 2019. Enhancing Commit Graphs with Visual Runtime Clues. In *Proceedings of the 2019 Working Conference on Software Visualization (VISSOFT)*. IEEE, Cleveland, USA, 1–11. https://doi.org/10.1109/VISSOFT.2019.00012

[14] Hagen Tarner, Valentin Bruder, Steffen Frey, Thomas Ertl, and Fabian Beck. 2022. Visually Comparing Rendering Performance from Multiple Perspectives. In *Proceedings of the 27th Symposium on Vision, Modeling, and Visualization (VMV)*. The Eurographics Association, Konstanz, Germany, 115–125. https://doi.org/10.2312/VMV.20221211

[15] Hagen Tarner, Veit Frick, Martin Pinzger, and Fabian Beck. 2020. Visualizing Evolution and Performance Metrics on Method Level as Multivariate Data. In *Proceedings of the 13th Seminar Series on Advanced Techniques & Tools for Software Evolution*, Eleni Constantinou (Ed.), Vol. 2754. CEUR-WS.org, Amsterdam, Netherlands (virtual). http://ceur-ws.org/Vol-2754/paper4.pdf