Volume-Based Large Dynamic Graph Analytics

Valentin Bruder*, Marcel Hlawatsch*, Steffen Frey*, Michael Burch†, Daniel Weiskopf*, and Thomas Ertl*
*University of Stuttgart, Germany
Email: firstname.lastname@visus.uni-stuttgart.de
†Eindhoven University of Technology, Netherlands
Email: m.burch@tue.nl

***Abstract*—We present an approach for interactively analyzing large dynamic graphs consisting of several thousand time steps with a particular focus on temporal aspects. We employ a static representation of the time-varying graph based on the concept of space-time cubes, i.e., we create a volumetric representation of the graph by stacking the adjacency matrices of each of its time steps. To achieve an efficient analysis of complex data, we discuss three classes of analytics methods of particular importance in this context: data views, aggregation and filtering, and comparison. For these classes, we present a GPU-based implementation of respective analysis methods that enable the interactive analysis of large graphs. We demonstrate the utility as well as the scalability of our approach by presenting application examples for analyzing different time-varying data sets.**

I. Introduction

The analysis and visualization of graph data [1], [2] is important for many application fields, e.g., to analyze social relationships, network traffic, or biological processes. The increasing amounts of data in many of these areas makes the visualization even more challenging [3]. This holds especially for dynamic graphs, which add another data dimension by incorporating time. Hence, visual analytics methods for dynamic graphs and networks are of increasing interest [4].

In this paper, we present a visual analytics approach [5], [6] for the visualization and analysis of large dynamic graphs. Our approach is especially targeted at the temporal analysis of graphs with several thousand time steps, e.g., tasks like detecting temporal patterns with respect to edges and clusters, or analyzing the temporal evolution of the graph structure. For this, we discuss three important classes of analytics methods (data views, aggregation and filtering, comparison) and present respective techniques. We base our techniques on a space-time cube representation [7] of the dynamic graph, where each time step of the graph is represented as an adjacency matrix. These matrices are then stacked to create a volumetric representation with time as the additional third dimension. This representation has the advantage that it is static and preserves the mental map [8], [9]. Furthermore, it is especially suitable for the temporal analysis of the graph since all time steps appear in an ordered sequence next to each other. To enable the interactive analysis of very large dynamic graphs with more than a thousand nodes and time steps, we use GPU-based methods for displaying different graph views, e.g., volume rendering for a volumetric graph representation.

The main contribution of this paper is the discussion of different classes of analytics methods for large dynamic graphs, along with respective methods. We implement and combine these methods in an integrated application, including a novel technique using the space-time cube metaphor and volume rendering, enabling interactive visualization of large dynamic graphs. Furthermore, our application features a technique to interactively compare arbitrary long sections of dynamic graphs against each other. We demonstrate the interplay and usefulness of the different classes of analytics methods, by means of two different scenarios: the analysis of a dynamic software call graph and flight connection data between US airports, both with more than a thousand time steps.

II. Related Work

Dynamic graphs. Visually analyzing a dynamic graph is a challenging task due to the continuously increasing size of graph data in all three relevant data dimensions: vertices, edges, and time. Various visualization and visual analytics techniques [5], [6] already exist that deal with dynamic graph data [4] focusing on either designing visual metaphors for displaying the dynamic graph in a scalable way or on computing graph structures evolving over time.

Traditional node-link diagrams are usually less suitable for showing graphs with a large number of time steps due to the display space required for a single time step. Therefore, derived techniques were developed that still use a node-link metaphor but with a different layout. Burch et al. [10] presented parallel edge splatting which arranges nodes on parallel axes resulting in small vertical stripes for each time step. This approach was later extended by Beck et al. [11] with a concept called rapid serial visual presentation, showing only a subset of the full graph in detail, to enable the visualization of more time steps. Burch et al. [12] developed another technique that reduces the required display space of parallel edge splatting by using overplotting of individual time steps.

However, all these approaches cannot solve the issue of crossing links leading to visual clutter that is inherent to node-link visualization. Therefore, other approaches are based on the adjacency list [13] or adjacency matrix [14] representation of a graph, which make dense graph structures displayable without visual clutter [15] caused by link crossings. Burch et al. [16] and Yi et al. [17] make use of matrix representations for dynamic graphs. They generate a global matrix that indicates the evolving weight as a line plot or color coded bar chart in each cell. Those techniques scale well with many time steps, but not necessarily with respect to many vertices and edges.

Techniques based on small multiples show adjacency matrices of dynamic graphs next to each other, generating a time-varying visualization of the data [18], [19]. Using such a visualization, a viewer has to jump back and forth to compare individual time slices and detect dynamic patterns in the data. Moreover, showing adjacency matrices side-by-side requires much screen space. Consequently, the visualization does not scale to thousands of time steps.

A third option to visualize dynamic graphs based on adjacency matrices can be obtained by stacking the matrices, which provides a good overview of evolving structures, if a suitable vertex clustering or ordering is applied [20], [21]. Stacking matrices generates a time-to-space mapping which guarantees a high degree of dynamic stability and consequently, supports the preservation of the mental map [8], [9]. Such a space-time cube representation can generate a visually scalable and interactive dynamic graph visualization, in which graph structures are aligned and can build longish shapes that are easily identifiable and selectable. This concept has already been successfully applied in the work of Bach et al. [22]. Schneider et al. [23] present a similar concept in their work, which they use for the exploration of dynamic structural connections in software components. Clustering and ordering techniques provide insights into the evolution of clusters. However, if the graph structure behaves chaotically over time, i.e., contains many alternating or oscillating dynamic patterns, a reasonable clustering cannot be easily computed.

Inspired by these works, we extended the space-time cube approaches to allow the analysis of larger data sets and further facilitate the analysis of large graphs by providing additional analysis and comparison methods.

Volume rendering. We use volume rendering as the basis of our dynamic graph visualization approach in this work. While raycasting induces a comparably high computational cost, it delivers not only high quality, but also high flexibility as each ray and even each sample along a ray may be individually adjusted as desired. Volume rendering is a classic field in visualization research that is typically used to directly render data from measurements (like CT scanners) or simulations. Since the availability of powerful and flexible graphics cards, GPU raycasting [24] has evolved as the de-facto standard technique in this context [25]. Beyond the classic application of raycasting to the visualization of a 3D volume, various approaches have been presented to map different types of data to a volume for rendering.

One particular active field in this context is the visualization of a time series of volumes in a single view. One approach is to interpret the data as a space-time hypercube and apply extended classic visualization operations like slicing, projection, or temporal transfer functions [7]. In this work, we also construct a space-time cube by stacking the time steps of (2D) adjacency matrices. For stacks of 2D data, Frey et al. [26] present an interactive visualization approach for detecting and exploring similarity in the temporal variation of field data, with a focus on studying periodic and quasi-periodic behavior. A popular approach to visualize multiple fields, is to combine them into a single value and then render the combined volume (e.g., Woodring et al. [27] use set operators for combination). Woodring and Shen [28] also proposed Chronovolumes, a static, direct rendering technique for time-varying data integrating all data over time into one volume. Balabian et al. [29] use temporal transfer functions and compositors to highlight areas of high change. We also use customized transfer functions in this work to enable the user to interactively highlight aspects of interest.

III. Background

In our visual analytics approach for large dynamic graphs, we especially focus on the analysis of temporal aspects and behavior. Therefore, we decided to use a static representation of the dynamic graph, because this provides numerous advantages compared to animation [30]. In the following, we describe our volumetric representation of dynamic graphs based on adjacency matrices, which is the core of our analytics approach.

A. Static Volumetric Graph Representation

In this work, we build on the concept of adjacency matrices. The rows and columns of an adjacency matrix denote the nodes of the graph. An entry in this matrix defines an edge between two nodes. In this paper, we use the following convention: An entry (i, j) at the i-th row and j-th column of the matrix denotes an edge from the node with index i to the node with index j. A dynamic graph is then represented by storing an adjacency matrix for every time step of the graph. Choosing the adjacency matrix representation has several reasons [20]:

1) It can show large graphs without inducing crossing or intersecting graphical elements, as it would be the case for traditional node-link diagrams.
2) We can generate a meaningful volumetric representation without any layout algorithm and naturally keep the same layout for every time step.
3) Adjacency matrices are well suited for revealing cluster structures in graphs.

Inspired by the work of Bach et al. [22] on dynamic graph visualization and space-time cube approaches [7] in general, we create a volumetric representation of the graph based on its adjacency matrices. These concepts have in common that they stack representations of individual time steps to a new data structure. In our case, adjacency matrices are 2D structures which are stacked to incorporate the temporal evolution of the graph. The resulting 3D matrix represents the full dynamic graph, where the x- and y-axes represent the nodes, entries in the plane defined by these axes represent edges (including their weights), and the z-axis represents time. The basic concept is illustrated in Figure 1.

B. Matrix Reordering

One issue of adjacency matrices is that their visual appearance strongly depends on the ordering of the nodes. A bad node order may result in rather noisy visual patterns with entries spread all over the matrix. A good node ordering shows cluster structures in the graph and similar columns or rows of the

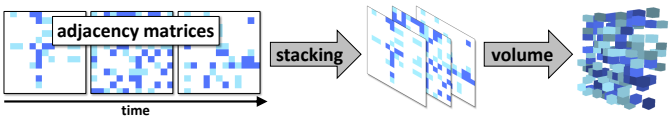


Fig. 1: In our static graph representation, 2D adjacency matrices for each time step of the dynamic graph are stacked to create a 3D volume, with the third dimension representing time.

matrix grouped next to each other. Therefore, matrix reordering algorithms play an important role when analyzing graphs with adjacency matrices. By using algorithms of the boost graph library [31], we employ three choices of sparse matrix ordering algorithms: Reverse Cuthill-McKee [32], King [33], and Sloan ordering [34]. An arbitrary number of consecutive adjacency matrices may be selected that are taken into account for the ordering by aggregating them beforehand.

IV. Classes of Analytics Methods

This work focuses on visualizing large dynamic graphs. However, while relying on animation for dynamic data is a popular choice, it has been shown to be ineffective as only a limited amount of information can be memorized by an observer [35]. Therefore, we used a (static) space-time representation for visualizing dynamic graphs as a basic design decision for our approach. Operating on this representation, we identified three important classes of techniques that are crucial for the analysis of large dynamic graphs (Figure 2), especially with respect to temporal features and behavior:

Data Views. Since large dynamic graphs usually exhibit a large amount of information covering different aspects of interest, it is important to offer different perspectives on the data using distinct views with suitable visualization techniques. In a complex analysis scenario, a single visualization is often not capable of revealing all interesting aspects.

Aggregation and Filtering. Directly showing large dynamic graphs with several hundreds to thousands of nodes and time steps typically results in visual clutter and overloaded visualizations. In particular, showing too much information that is not necessarily relevant to a specific question, may obfuscate the information of interest. Filtering and aggregation of data are useful approaches to reduce such issues.

Comparison. The analysis of a single complex dynamic graph is already challenging in itself. When two or more different dynamic graphs need to be compared, this typically becomes a difficult problem to solve using only the methods discussed so far. Therefore, it is important that comparison is supported with specifically designed visualizations.

We implemented different methods for these building blocks in our graph analytics system (see Figure 2). In the following, we explain these methods for the different classes in detail, and discuss how they support the analysis of dynamic graphs.

A. Data Views

Depending on the use case and the problem setting, there might be many different aspects of the graph data that are of

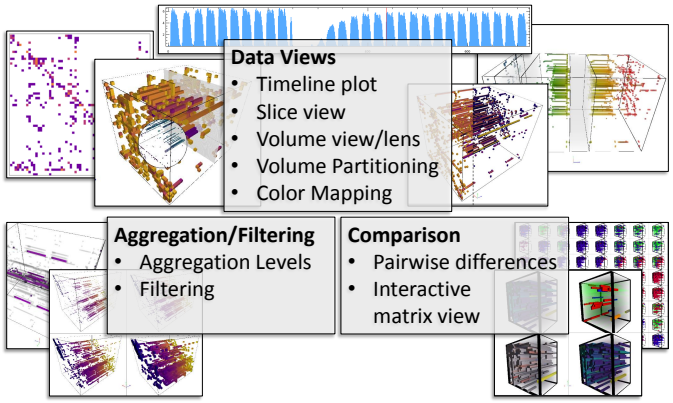


Fig. 2: Three important classes of analytics methods for large dynamic graphs. We implemented and discuss respective techniques, and show their utility with application examples.

relevance. Usually, there is not a single visual representation that can reveal all these aspects. Furthermore, different types of visualization techniques might be better suited for different analysis and exploration tasks. Therefore, we offer multiple distinct data views to the analyst that reveal different aspects of the data.

Timeline Plot. We implemented a 2D plot visualization showing different graph metrics on a timeline (Figure 3a), e.g., edge count or matrix attributes such as linear arrangement. This visualization is well-suited as an overview and to reveal global temporal patterns, e.g., if the density of the whole graph changes over time, it is visible in the edge count plotted over time. Besides showing relevant graph properties, this plot is also a suitable interface for different interaction and selection tasks. For instance, the timeline allows us to select individual time steps, which are also selected or marked in the other views. Furthermore, the timeline view can also be used to create and manipulate the split marks for the volume partitioning (see discussion of volume partitioning below).

Slice View. The previously described timeline plot can only show strongly aggregated graph properties, such as the number of edges in each timestep. However, it is not possible to see details of the graph structure there. To provide information about individual time steps or nodes of the graph, we implemented several 2D slice views of the graph volume (Figure 3b). They allow for an analysis of single time steps or single nodes, by projecting graph properties on a plane (e.g., single or aggregated time steps).

Volume View. With the slice views, it is possible to analyze single time steps or nodes. However, analyzing the graph structure and the temporal evolution on a global level is difficult because it requires to inspect all slices and relate them to each other. To achieve this, we also include a direct visualization of the complete graph volume (Figure 3c). It provides two viewing modes. The first one is a 2×2 tiled view that allows to see and compare different graph properties or aggregation levels of the data (see discussion of aggregation levels in Section IV-B below). If desired, this view can be changed to

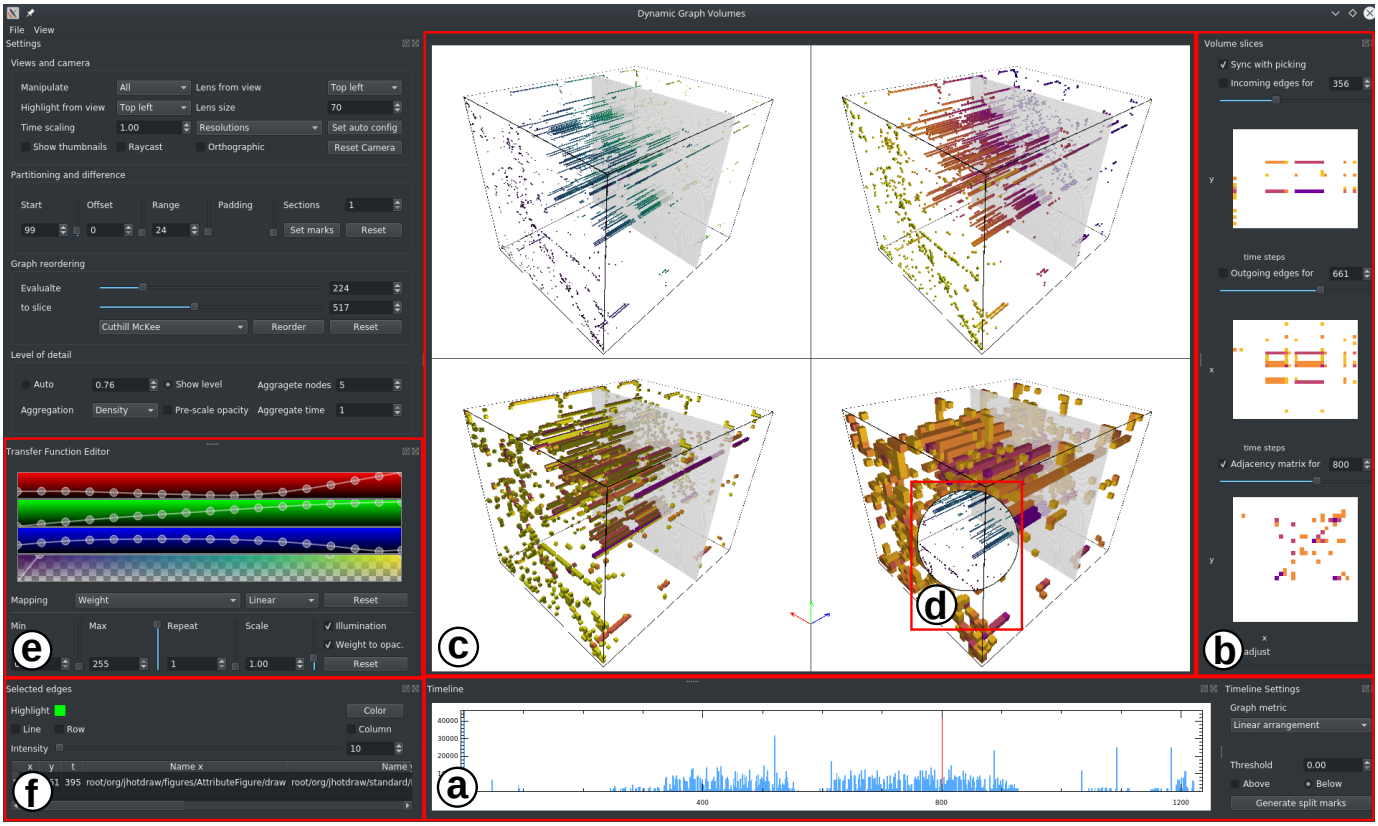


Fig. 3: The GUI of our dynamic graph analysis tool, consisting of (see Section IV-A): (a) timeline plot, (b) 2D slice views, (c) volume view, (d) volume lens, and (e) color map editor. The list view (f) shows information of selected edges and nodes.

a $1 + 3$ view mode with one large view in the center and three smaller views in the corners. The large view enables a more efficient exploration of the data, while the smaller views provide additional context. By simply clicking on the smaller views, the selected one is enlarged and now shown in the center. Each view has one dedicated parameter set defining the visualization, i.e., every view can show the data with a different aggregation level and color mapping. To support comparison tasks, camera parameters of all views are synchronized. With this functionality, different properties of the graph can be shown at the same time, providing a more comprehensive view on the data.

Volume Lens. Apart from the possibility to use the four different views to see various aspects of the data, we also provide a mouse-controlled lens (Figure 3d). The lens shows the graph data in the volume view with different visualization parameters, such as aggregation level and/or color mapping. For instance, it is possible to show the full graph with a high aggregation level, to better convey the overall graph structure, while inside the lens, a lower level is shown to provide a view on the non-aggregated graph data. Another example for the usage of the lens is to show the edge weights (via color mapping) inside, while the lifetime of the edges is shown outside the lens.

Volume Partitioning. The main focus of our visual analytics approach for large dynamic graphs lies on the analysis of temporal aspects. To improve our interaction possibilities in that regard, we integrate the ability to partition the volume into sub-volumes along the temporal dimension (see Figure 4). Here, each sub-volume represents a certain time span of the graph. By showing only one respective sub-volume, the problems of visual clutter and occlusion of the graph volume view are reduced. The volume partitioning is defined by setting split marks in the timeline plot (Figure 3a). When setting split marks, the volume gets visually separated in the main view by introducing gaps between sub-volumes. A user can interactively add, delete, and move the split marks and thereby adapt the partitioning. We also offer the option to let our tool automatically set the split marks based on the graph metric shown in the timeline. For this, the analyst defines a threshold, and whenever the metric exceeds this threshold, a split mark is set automatically.

Color Mapping. We visualize certain properties of the dynamic graph by mapping them to color, e.g., the weights of the edges. We offer further modes that support the analysis of the temporal behavior of the graph (see Figure 5), which can be selected using the color map editor (Figure 3e). Mapping the temporal dimension to color allows for a better comparison of the temporal relation of edges, e.g., how long do certain edges exist. Although this information is also visible in the third dimension of the volume, the projection to 2D makes a comparison potentially difficult. The usage of color to explicitly represent this aspect, mitigates this limitation. Further quantities

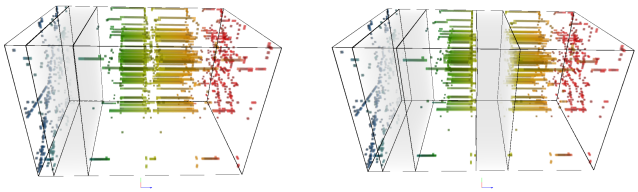
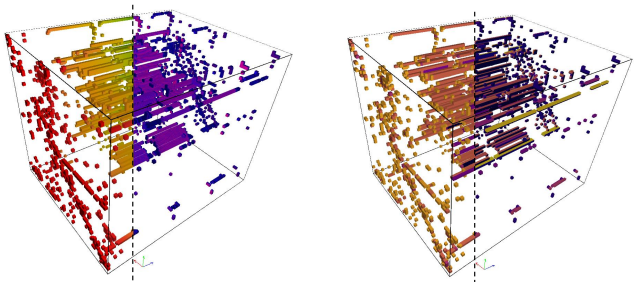


Fig. 4: It is possible to split the graph volume into several sub-volumes, representing different time ranges.



(a) time/edge count (b) edge weight/edge lifetime

Fig. 5: Different visual mappings of dynamic graph properties: Color mapping of time ((a), left), edge count per time step ((a), right), edge weights ((b), left), the time span each edge exists in the dynamic graph ((b), right).

that can be shown with color mapping include, among others, edge count and edge lifetime, which indicates the total time an edge exists in the full graph.

Note that, as we use a static (volumetric) representation for our dynamic graph, the color mapping we employ is implemented via the well-known concept of transfer functions from traditional volume rendering. In that regard, it also bears some similarity to the filtering step discussed in Section IV-B (color mapping defines color, while filtering steers opacity).

B. Aggregation and Filtering

A direct visualization of the complete graph is usually not suitable for a detailed analysis. It might be even difficult to obtain a good overview of the graph, since displaying several millions of edges results in a very cluttered visualization. Therefore, it is important to reduce the amount of data presented to the user. We offer two different approaches for this task, which are discussed below: aggregation and filtering.

Aggregation. Individual elements can be very small, especially on the overview level. As a result, there is a high risk that entries are not visible due to projection and rasterization issues. Furthermore, aliasing artifacts such as Moiré patterns may occur. Therefore, we offer several methods for aggregating the data during a down-sampling process that we employ to create different resolution levels. Depending on the used aggregation method, different features in the data are visible. In our implementation, we offer the following aggregation methods: min, max, average weight, and density of edges. Aggregation along the time axis is optional, i.e., not only neighboring edges but also time steps are aggregated if desired. In this case, a graph voxel represents multiple edges and time steps (see Figure 6).

Choosing the right aggregation level depends on the respective analysis task and the properties of the graph, e.g., how sparse it is. Therefore, we let the analyst interactively switch between different aggregation levels. To ease the selection of a suitable level, we initially present four different levels in the main view. The analyst can then select the desired level by clicking on the respective view. Furthermore, two aggregation levels can be combined in the same view by using our lens (see Section IV-A), which allows showing the graph volume with a different aggregation level in the same view.

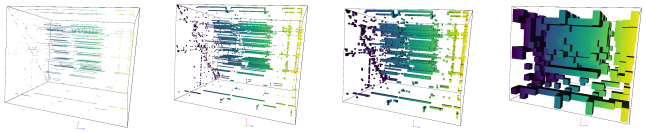
Filtering. To reduce visual overload and occlusion, it can be very helpful to filter out edges with specific properties, e.g., the ones featuring low weights. This allows to focus on certain aspects of the graph, e.g., edges with high weights. By changing transparency, it is possible to flexibly explore different aspects of the data. Technically, this is implemented as transfer function mapping of our volumetric graph representation (as discussed in Section IV-A).

C. Comparison

In addition to the volume partitioning and selection, we provide another modality to analyze temporal aspects of dynamic graphs by offering a graph difference visualization (see Figure 9). Essentially, the analyst can select the starting point, size, and the number of time steps that are compared. Based on this selection, a visualization of pairwise differences is presented in a matrix layout. The original graph data is shown on the diagonal and the differences are shown in the upper respective lower triangle of the matrix. The upper one shows the comparison of the existence of edges (i.e., new edge appeared/disappeared or stayed), and the lower triangle shows the difference between weights. All views are encoded with color mapping, allowing to highlight or filter respective differences. Since slight offsets between time ranges can result in huge differences, e.g., in case of repeating patterns, our application allows the analyst to interactively change the comparison parameters and instantaneously see the outcome of the comparison.

V. Implementation

We run the computation-intensive parts of our visualization approach on the GPU, to enable the interactive exploration and analysis even for large dynamic graphs (potentially featuring thousands of nodes and time steps). Those parts naturally include the volume view (rendered with raycasting techniques), but also the slice views and generation of the aggregation levels. Matrix reordering is done in parallel on the CPU. For the volume view, we transfer the stacked adjacency matrices as a 3D-texture to GPU memory and perform further processing of the data and generation of the visualization on the GPU using OpenCL. While our current implementation technically



(a) Level 1 (b) Level 3 (c) Level 4 (d) Level 6

Fig. 6: Different aggregation levels (using edge density) of a software call graph. By showing different levels at the same time, an analyst can choose a level that provides a good balance of visibility and coarseness.

limits the size of graph data sets to the available GPU memory, the use of out-of-core techniques could mitigate this limitation.

For rendering the stacked adjacency matrices, the analyst may choose between two modified front-to-back raymarching algorithms with early ray termination. The first is an implementation of the 3D digital differential analyzer algorithm [36], with local lighting based on actual voxel surfaces. This method and lighting enables an analyst to more easily distinguish between single or isolated voxels and opaque structures, making it better suited for viewing sparse graphs. The second rendering method is based on standard raycasting, i.e., we sample the generated 3D-texture with a constant step size along the rays. In that case, we calculate local lighting based on the gradient of each voxel, using central differences. This rendering and lighting approach is more suited to display density distributions, especially in the case of dense graphs and the usage of transparency.

During raycasting, we map the sampled density values to color and opacity (through color mapping and filtering) and finally composite them into pixel colors for display. We use raycasting as the basis for our 3D visualization approach because it has several advantages compared to rendering geometry directly. Regarding performance, volume raycasting enables us to render millions of dedicated matrix entries at the same time while keeping interactive frame rates. Compared to rendering geometry, we do not have to perform front to back ordering and can easily use transparency for various visual purposes, such as highlighting items or filtering edges with specific properties. Additionally, we can display multiple views with the same camera setup in a single viewport without generating a significant overhead. Using our application on a workstation equipped with an NVIDIA TitanX (Pascal) GPU, an Intel Core i7-6700 and 16GB of RAM, we were able to render data sets of more than 1000 nodes and 8000 timesteps (possibly over eight billion edges) interactively, i.e., with more than 20 FPS on average. Note that volume renderings of dense graphs are typically generated faster than sparse ones, due to our usage of early ray termination. However, this highly depends on the used opacity mapping as well.

VI. Application Examples

We demonstrate our approach with two different time-dependent graph data sets: a flight data set in which the edges represent the connections between airports (Section VI-A), and a dynamic call graph representing function calls in a Java application (Section VI-B). Based on these examples, we discuss the suitability of the different classes of analytics methods (see Section IV) for different tasks in Section VII.

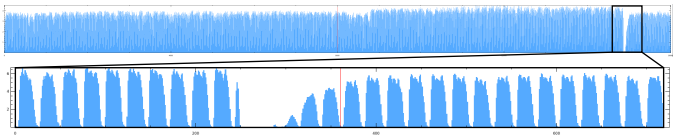


Fig. 7: Timeline plot of the flight data set showing the number of edges per time step. The upper image shows the complete data set, while the lower one shows data of September, 2001.

A. Flight Connections

We first look at a data set representing the US domestic flight traffic in the years 2000 and 2001. The dynamic graph contains 16,000 time steps and 234 nodes. The number of edges per time step is shown in the timeline plot in Figure 7. Immediately noticeable is a big gap in the graph after the 9/11 terrorist attacks. The reason for this is that the complete air traffic in the US was shut down for around one day after the attacks. Air traffic recovered in the following days. Besides this large gap at 9/11, we can see a regular pattern of small gaps, which occur on a daily basis in the data set. The reason is that there are several hours each night, when hardly any or no air traffic occurs at all.

Figure 8 shows the volume view of the flight data set, with the color map showing time. To reduce clutter and aliasing artifacts and to reveal the temporal patterns in the graph, we show aggregation level 2 in the visualization, i.e., aggregating 4 neighboring nodes (the time dimension is not aggregated), by averaging the weights. Furthermore, the matrix has been reordered using the Cuthill-McKee algorithm (based on the first 24 hours of the data set) and the volume was split in the middle of the time range. The visualization shows that there is an increase of the edge count in the second year (right sub-volume). From the structures in the adjacency matrix, we can also see that the new edges appeared at the boundary of the matrix. Investigating the data more closely, it shows that some airports do not have any flights in the year 2000. We assume that information from these airports were not recorded before 2001 or new flights have been added.

With our volume lens functionality, we can overlay the visualization with a different representation of the graph (Figure 8b). In this case, a different color mapping is shown to highlight edges with high weight. The edge weight corresponds to the number of flights per hour on the same route. We can see that several flights of the same route during one hour mainly occur at the large airports that have connections to many others, which are placed in the lower right corner by the reordering.

In Figure 9, we demonstrate the comparison of different time spans with our graph difference visualization. In this case, we extracted a dynamic graph for each day from the complete graph. We then compare eight consecutive time spans of 24 hours, in order to extract similarities and differences

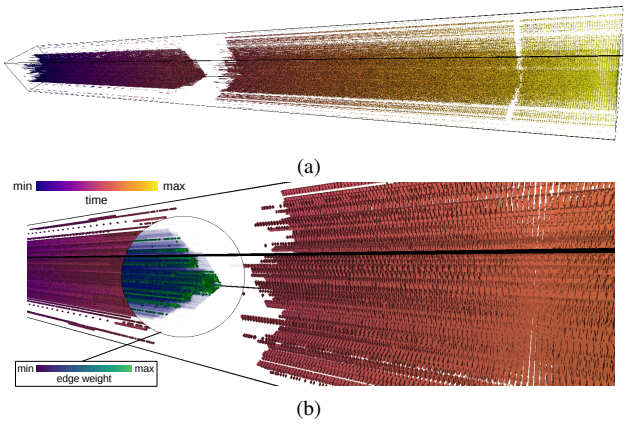


Fig. 8: (a) Volume visualization of the complete flight data set with color showing time. The volume was split at half of the time range. (b) Close-up of the split area with the lens overlaying a different color mapping showing the edge weight.

in the flight connections of different days. On the overview, showing a tiled matrix of differences for all days, the color mapping reveals on a coarse level between which days large differences or similarities occur. However, details are hardly visible, e.g., if there are big changes in the flight connections of a specific airport. For such detailed analysis, we can enlarge the comparison of two different days. Here, we compare the first two days. We can observe that most connections are stable. However, on Sunday, there are more flights in the morning compared to the following Monday, where more flights are scheduled in the evening.

B. Software Call Graph

As a second application, we analyze a dynamic software call graph (that has previously also been used by Beck et al. [11]). This is a dynamic graph containing 982 nodes, $32,259$ weighted and directed edges, and $1,077$ time steps. The call graph was obtained using a Java profiling tool and represents the function calls during the execution of a drawing application. The graph captures the following steps during execution: program start, creation of a new document, drawing a rectangle and a circle, and finally writing text into the circle. The edge weights are the execution times of the respective function calls represented by the edges in the graph. Matrix reordering is not necessary as the initial ordering is already based on the hierarchy of the software project which provides respective semantics.

We start our analysis on the overview level (see Figure 10), using our volume view in four different aggregation levels. We select the level in the lower left corner for further inspection. In this level, the graph structure is clearly visible without hiding too many details due to the accumulation. Looking at the coarser aggregation level (lower right corner), we can observe that many details are lost, e.g., it is hard to see that there are two similar blocks covering longer time spans in the center.

Continuing the analysis with the selected aggregation level, we can get a general overview of the graph structure and its temporal evolution. Using this data view, we are able to visually structure the temporal evolution into several blocks of similar behavior. Using our volume partitioning implementation, we can automatically divide the volume into several sub-volumes based on different graph metrics. In this case, we use the edge count of each time step to partition the volume. Our tool allows us to interactively adapt the partitioning by changing, deleting, or adding split marks. In this example, the automatic selection works well and only slight modifications are required to structure the data into four regions of different behavior: In the first time span, there are many edges that exist only for a short period of time. Then, there are two blocks with several long-lasting edges. At the end, the edge count increases again. However, these edges exist only for a short time period. Comparing this to the different phases of the program execution (start, drawing, writing), we can draw several conclusions: Starting the program and creating a new document requires a lot more function calls than drawing the different objects. However, this requires less execution time. Drawing objects in the second and third phase results in constantly calling the same functions for a longer time span. Writing text at the end results again in more function calls for a shorter time span.

The next step in our analysis is the detailed exploration of these different time spans. For this, we select the respective sub-volume, which is now shown in the large center view. We first use the slice view (see Section IV-A) to get an aggregated view of the edges over time. Interesting graph structures are visible, e.g., there are almost no edges in the upper right half of the matrix. The diagonal of the matrix is densely covered, i.e., many functions are calling neighboring functions in the hierarchy of the software project. Furthermore, we see an almost complete line in the center right area, which means that this function calls many other functions, i.e., this function might be crucial for the application. Looking up some of these function calls in the list view reveals that the function “createTools” calls different “init” functions. We assume that this function is responsible for creating the drawing tool bar. Rotating the sub-volume in the volume view reveals that the highlighted function calls only appear in the first time step, i.e., these functions are called only once in this time range.

After examining the first time span, we continue our analysis with the next sub-volume. After selecting the time span via the timeline plot, the respective sub-volume is now shown in the volume view in the center. As we have already seen in the overview, the graph structure is quite different compared to the first block. The edge count is rather low, but the duration of most of the edges is quite long. Furthermore, these edges seem to have a similar duration. We can see from the position in the matrix volume that some of the calls are from the same function or the same functions are called, i.e., they are in the same column respectively row. While all these function calls have similar durations in the selected time range, it might be of interest to see whether some of these calls occur more often than others with respect to the full time range of the graph. Therefore, we display the global edge life time with the respective color mapping. Now we can see which functions

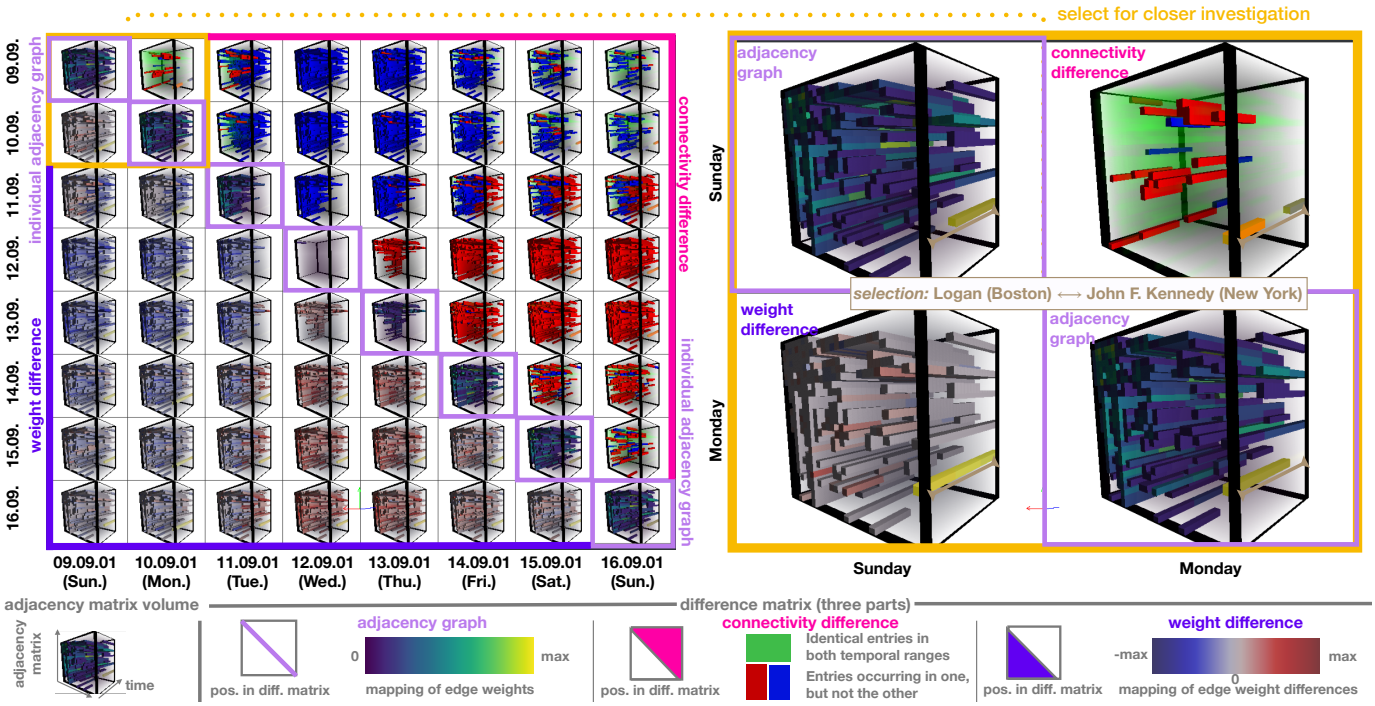


Fig. 9: *Left:* Graph difference visualization to investigate temporal changes in dynamic graphs. Here (see Section VI-A), we depict connections between US airports in September 2001 (diagonal), along with differences between days (upper right and lower left). *Right:* Changes in connections from a Sunday to the following Monday in an enlarged difference view. Flight schedules are largely the same (indicated in green), yet there are predominantly less flights in the early morning on Sunday, but more in the afternoon, e.g., flights between Boston Logan and New York JFK (highlighted yellow/orange).

are called most often during the execution of the application. In this case, they are shown in yellow based on the color map. We select one of these edges to analyze it (marked area). The list view tells us that this function is responsible for redrawing the background of the application. From this information, we derive that the redraw function is called every time something changes in the GUI of the application.

Proceeding with the third block, we can see similar structures as in the previous block. Therefore, we want to compare this block with the previous block by using our graph difference visualization (see Section IV-C). As we can see there, almost the same edges occur in both time spans with the same weights, in this case representing the execution times. There are only a few different edges in the center area. By selecting these edges and using the list view, we can find out that function calls to "RectangleFigure/displayBox" disappear and get replaced by function calls to "EllipseFigure/displayBox". Hence, drawing different graphical objects require almost the same function calls, only a different "displayBox" is used.

VII. Discussion of Analytics Classes

In the following, we discuss the roles of the classes of analytics methods in the analysis process, based on the examples presented above.

Providing different data views is crucial to gain insight into large dynamic graph data, as can be seen in the case of the investigated flight data set, which contains more than 16,000 time steps. The timeline plot shows the general evolution of the number of flight connections (edge density), however, details about the individual connections are not visible. Those are provided in the volume view that shows the structure of edges between nodes as well. However, this view may exhibit occlusion and visual clutter, in particular for more complex graph data. In this case, aggregation and filtering is required to remove some of the details on the overview level. These details can be reintroduced with the interactive lens, which shows a different data view or aggregation level in a user-specified area. An (aggregated) slice view can be suitable for this task as well. Furthermore, it is possible to extract detailed information of the graph data by selecting individual elements. In case of the flight data, they represent airports (nodes) or connections (edges), e.g., the user may obtain the information which airports have a high number of connections. Using these different data views and aggregation levels, a comparison of different parts of the graph would still be tedious, e.g., comparing the flight connections of different days. For this task, we provide the graph difference visualization that can visually reveal which connections remain and which change between days.

In the case of the call graph application example, volume partitioning is a particularly effective analytics method, due to the four distinct time ranges with different behavior. This allows us to select certain sub-regions of the data (filtering), in this case time ranges, to continue the analysis with a reduced

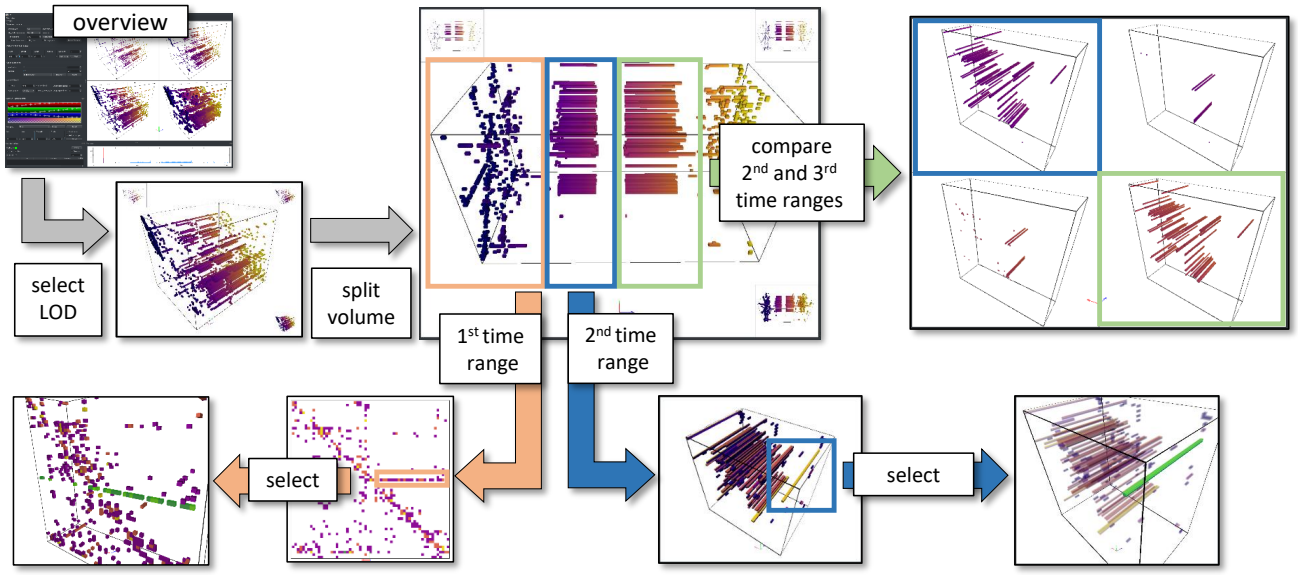


Fig. 10: Application of our visual analytics approach to the call graph data set. Starting on the overview, an aggregation level is chosen for analyzing the temporal evolution of the graph. The data is divided in four temporal regions based on the overall structure. These regions are then analyzed in detail, going from overview level down to individual edges. Edges with interesting behavior are selected and analyzed. The second and third region are compared with our graph difference visualization.

amount of data. However, it can also help using an aggregated representation of the data. Due to the large number of nodes, a direct visualization of the unaggregated data would result in very small structures in the visualization. After selecting a certain time range of the data, again, different data views can be used to extract information. Here, the aggregated slice view can be used to gain additional insight. For instance, which functions call many other functions, or a suitable color mapping can be applied to highlight functions that are called most often. In the example data, the high similarity of the second and third time range makes it especially difficult to see the differences between both time ranges. Again, this is efficiently revealed by our graph difference visualization.

With both examples, we have demonstrated the importance of the different classes of analytics methods (see Section IV). Since every visualization technique has its limitation, it is important to provide different data views. For example, the volume view provides a visualization of the complete graph but is prone to occlusion and visual clutter. To analyze the evolution of the edge count, the timeline plot is often the more suitable visualization. But even with a visualization view that is adequate for a specific task, the size of a large dynamic graph may still lead to a visualization that is difficult to read. Therefore, aggregation and filtering are crucial to be able to focus on specific parts or attributes of the data. Finally, comparing different graphs or various time spans of a single dynamic graph is an even more complex task. To efficiently handle respective tasks, we directly support them with a dedicated graph difference visualization.

VIII. Conclusion and Future Work

We presented a visual analytics approach that allows for interactive analysis of large dynamic graphs consisting of several thousand timesteps. The core of our approach consists of three classes of analytics methods that we identified as being of particular importance for efficiently analyzing large dynamic graphs: data views, aggregation/filtering and comparison. We implemented and combined these classes in an interactive visual analytics application that we employ to demonstrate the usefulness and applicability in the form of two distinct application scenarios.

Our implementation features a GPU-based volume view that provides the user with an overview of the graph data structure, and that is augmented by additional views and projections of the data. Aggregation is supported in the different views, while filtering and highlighting can be done by adapting the color and/or opacity mapping. Finally, we directly support the temporal comparison of different time spans or multiple dynamic graphs with an integrated graph difference visualization. Our application examples demonstrated not only the scalability of our approach to dynamic graphs with several thousand time steps, but also show how to temporally analyze and compare different time ranges of a dynamic graph.

One important goal for future work is to further increase the capabilities of our tool with respect to the number of nodes and time steps. Out-of-core concepts might allow the visualization and analysis of graph data sets that are ten to one hundred times larger than the ones currently supported. Another direction of future work is to improve the automatic determination of suitable visualization parameters, e.g., the aggregation level and the reordering method.

ACKNOWLEDGMENT

The authors would like to thank the German Research Foundation (DFG) for supporting the project within project A02 of SFB/Transregio 161.

REFERENCES

[1] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[2] M. Kaufmann and D. Wagner, Eds., *Drawing Graphs, Methods and Models*, ser. Lecture Notes in Computer Science. Springer, 2001, vol. 2025.

[3] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J. Fekete, and D. W. Fellner, "Visual analysis of large graphs: State-of-the-art and future research challenges," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1719–1749, 2011.

[4] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "A taxonomy and survey of dynamic graph visualization," *Computer Graphics Forum*, vol. 36, no. 1, pp. 133–159, 2017.

[5] D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler, "Visual analytics: Scope and challenges," in *Visual Data Mining - Theory, Techniques and Tools for Visual Analytics*. Springer, 2008, pp. 76–90.

[6] J. Thomas and J. Kielman, "Challenges for visual analytics," *Information Visualization*, vol. 8, no. 4, pp. 309–314, 2009.

[7] B. Bach, P. Dragicevic, D. Archambault, C. Hurter, and S. Carpendale, "A descriptive framework for temporal data visualizations based on generalized space-time cubes," in *Computer Graphics Forum*, vol. 36. Wiley Online Library, 2017, pp. 36–61.

[8] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Journal of Visual Languages and Computing*, vol. 6, no. 2, pp. 183–210, 1995.

[9] D. Archambault, H. C. Purchase, and B. Pinaud, "Animation, small multiples, and the effect of mental map preservation in dynamic graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 4, pp. 539–552, 2011.

[10] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf, "Parallel edge splatting for scalable dynamic graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2344–2353, 2011.

[11] F. Beck, M. Burch, C. Vehlow, S. Diehl, and D. Weiskopf, "Rapid serial visual presentation in dynamic graph visualization," in *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Sept 2012, pp. 185–192.

[12] M. Burch, M. Hlawatsch, and D. Weiskopf, "Visualizing a sequence of a thousand graphs (or even more)," *Computer Graphics Forum*, vol. 36, no. 3, pp. 261–271, 2017.

[13] M. Hlawatsch, M. Burch, and D. Weiskopf, "Visual adjacency lists for dynamic graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 11, pp. 1590–1603, 2014.

[14] M. Ghoniem, J. Fekete, and P. Castagliola, "On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis," *Information Visualization*, vol. 4, no. 2, pp. 114–135, 2005.

[15] R. Rosenholtz, Y. Li, J. Mansfield, and Z. Jin, "Feature congestion: a measure of display clutter," in *Proceedings of the 2005 Conference on Human Factors in Computing Systems, CHI*, 2005, pp. 761–770.

[16] M. Burch, B. Schmidt, and D. Weiskopf, "A matrix-based visualization for exploring dynamic compound digraphs," in *Proceedings of 17th International Conference on Information Visualisation, IV*, 2013, pp. 66–73.

[17] J. S. Yi, N. Elmqvist, and S. Lee, "Timematrix: Analyzing temporal social networks using interactive matrix-based visualizations," *International Journal of Human-Computer Interaction*, vol. 26, no. 11&12, pp. 1031–1051, 2010.

[18] A. Perer and J. Sun, "Matrixflow: temporal network visual analytics to track symptom evolution during disease progression," in *AMIA annual symposium proceedings*, vol. 2012. American Medical Informatics Association, 2012, p. 716.

[19] B. Bach, N. H. Riche, T. Dwyer, T. M. Madhyastha, J. Fekete, and T. J. Grabowski, "Small multipiles: Piling time to explore temporal patterns in dynamic networks," *Computer Graphics Forum*, vol. 34, no. 3, pp. 31–40, 2015.

[20] M. Behrisch, B. Bach, N. H. Riche, T. Schreck, and J. Fekete, "Matrix reordering methods for table and network visualization," *Computer Graphics Forum*, vol. 35, no. 3, pp. 693–716, 2016.

[21] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[22] B. Bach, E. Pietriga, and J.-D. Fekete, "Visualizing dynamic networks with matrix cubes," in *CHI Conference on Human Factors in Computing Systems*, 2014, pp. 877–886.

[23] T. Schneider, Y. Tymchuk, R. Salgado, and A. Bergel, "Cuboidmatrix: Exploring dynamic structural connections in software components using space-time cube," in *Software Visualization (VISSOFT), 2016 IEEE Working Conference on*. IEEE, 2016, pp. 116–125.

[24] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A simple and flexible volume rendering framework for graphics-hardware-based raycasting," in *Proceedings of the Fourth Eurographics / IEEE VGTC Conference on Volume Graphics*, ser. VG'05. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2005, pp. 187–195.

[25] M. Hadwiger, P. Ljung, C. R. Salama, and T. Ropinski, "Advanced illumination techniques for gpu volume raycasting," in *ACM SIGGRAPH ASIA 2008 Courses*, ser. SIGGRAPH Asia '08. New York, NY, USA: ACM, 2008, pp. 1:1–1:166.

[26] S. Frey, F. Sadlo, and T. Ertl, "Visualization of temporal similarity in field data," *IEEE Vis. Comput. Gr.*, vol. 18, pp. 2023–2032, 2012.

[27] J. Woodring and H.-W. Shen, "Multi-variate, time varying, and comparative visualization with contextual cues," *IEEE transactions on visualization and computer graphics*, vol. 12, no. 5, pp. 909–916, 2006.

[28] ——, "Chronovolumes: a direct rendering technique for visualizing time-varying data," in *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume Graphics*, ser. VG '03. New York, NY, USA: ACM, 2003, pp. 27–34.

[29] J.-P. Balabanian, I. Viola, T. Möller, and E. Gröller, "Temporal styles for time-varying volume data," in *Proceedings of 3DPVT'08 - the Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, S. Gumhold, J. Kosecka, and O. Staadt, Eds., June 2008, pp. 81–89.

[30] B. Tversky, J. B. Morrison, and M. Betrancourt, "Animation: can it facilitate?" *International Journal of Human-Computer Studies*, vol. 57, no. 4, pp. 247–262, 2002.

[31] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.

[32] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*. ACM, 1969, pp. 157–172.

[33] I. P. King, "An automatic reordering scheme for simultaneous equations derived from network systems," *International Journal for Numerical Methods in Engineering*, vol. 2, no. 4, pp. 523–533, 1970.

[34] S. Sloan, "An algorithm for profile and wavefront reduction of sparse matrices," *International Journal for Numerical Methods in Engineering*, vol. 23, no. 2, pp. 239–251, 1986.

[35] A. Joshi and P. Rheingans, "Illustration-inspired techniques for visualizing time-varying data," in *Visualization, 2005. VIS 05. IEEE*, 2005, pp. 679–686.

[36] J. Amanatides, A. Woo *et al.*, "A fast voxel traversal algorithm for ray tracing," in *Eurographics*, ser. 87, no. 3, 1987, pp. 3–10.