informatics

MDPI

Article

AVIST: A GPU-Centric Design for Visual Exploration of Large Multidimensional Datasets

Peng Mi [1,*], Maoyuan Sun [2], Moeti Masiane [1], Yong Cao [3] and Chris North [1]

[1] The Computer Science Department, Virginia Tech, Blacksburg, VA 24060, USA; moeti@cs.vt.edu (M.M.); north@cs.vt.edu (C.N.)
[2] Department of Computer and Information Science, University of Massachusetts Dartmouth, Dartmouth, MA 02747, USA; smaoyuan@umassd.edu
[3] The Boeing Company, 3455 Airframe Dr, North Charleston, SC 29418, USA; claio0128@gmail.com
* Correspondence: mipeng@cs.vt.edu; Tel.: +1-540-838-5474

Academic Editor: Olga Kurasova
Received: 31 August 2016; Accepted: 28 September 2016; Published: 7 October 2016

Abstract: This paper presents the Animated VISualization Tool (AVIST), an exploration-oriented data visualization tool that enables rapidly exploring and filtering large time series multidimensional datasets. AVIST highlights interactive data exploration by revealing fine data details. This is achieved through the use of animation and cross-filtering interactions. To support interactive exploration of big data, AVIST features a GPU (Graphics Processing Unit)-centric design. Two key aspects are emphasized on the GPU-centric design: (1) both data management and computation are implemented on the GPU to leverage its parallel computing capability and fast memory bandwidth; (2) a GPU-based directed acyclic graph is proposed to characterize data transformations triggered by users' demands. Moreover, we implement AVIST based on the Model-View-Controller (MVC) architecture. In the implementation, we consider two aspects: (1) user interaction is highlighted to slice big data into small data; and (2) data transformation is based on parallel computing. Two case studies demonstrate how AVIST can help analysts identify abnormal behaviors and infer new hypotheses by exploring big datasets. Finally, we summarize lessons learned about GPU-based solutions in interactive information visualization with big data.

Keywords: big data; interactive data exploration and discovery; multidimensional dataset; GPU

1. Introduction

As data sizes increase, interactive visual analysis becomes a key for gaining usable insights from massive data. To support sensemaking of big data, three aspects are emphasized in exploratory visual analysis. First, flexible data filtering (e.g., cross-filtering [1,2]) can empower analysts to "prune" large datasets to gain buried relationships. Second, fine data details should be provided on demand, so that analysts can capture certain subtle abnormal activities. Last, exploratory data analysis is usually an interactive and iterative process. Thus, big data exploration needs high performance computing to provide real-time frame rate performance. For example, when cyber security analysts attempt to identify attacks from millions of records in network traffic logs, they have to filter data records iteratively to finally narrow down to suspicious records.

However, there are many challenges to interactive analysis of big data. When data become bigger, they cannot be loaded into memory or the computation speed is not sufficient for real-time performance. In order to feed more data into memory, Tableau [3] introduces Tableau Data Engine (TDE) [4]. The TDE highlights a specialized column-oriented data format, which supports a high data compression ratio (considering repeating values in the columns) and provides high level data aggregation information. To achieve fast visual querying of big data, imMens (a system designed

to support interactive visual exploration of large data sets) [5] utilizes GPU resources to speed up performance. It aggregates data into data tiles using the data cube technique [6] and uses WebGL for data processing and rendering to improve performance. However, the data cube technique suffers from the data scalability problem, where data grow exponentially with the dimensionality. In summary, both Tableau and imMens provide the capability to explore high level aggregated information of big data.

Different from these, our paper aims to flexibly investigate fine-grained data details, such as identifying buried data relationships by complex filtering. Additionally, we propose a new data processing pipeline integrated with GPUs to achieve this goal. As a proof of concept, we implement an Animated VISualization Tool (AVIST), which can analyze millions of multi-dimensional data records per second on an off-the-shelf desktop computer. We emphasize "animation" in the context of time series multidimensional datasets to help identify temporal data patterns. In summary, our key contributions are as follows:

(1) We propose a GPU-centric design to support interactive visual exploration of large datasets, which emphasizes the use of GPU for data management and computation.
(2) We design a data dependency graph to characterize GPU based data transformations, which supports data aggregation and visualization on demand.
(3) We implement AVIST following our GPU-centric design as a proof-of-concept. AVIST features animation and cross-filtering interactions to slice big data into small data and GPU parallel computing to transform raw data into visual primitives.
(4) We present two usage scenarios to demonstrate that AVIST can help analysts identify abnormal behaviors and infer new hypotheses by visual exploration of big datasets.
(5) We discuss lessons learned about the application of GPU methods to address performance challenges in interactive, visual exploration of big data.

2. Related Work

Our survey includes four aspects about visual exploration of big data: (1) which kind of data should be visualized and what are their characteristics; (2) how does one store and manage big data; (3) how does one compute big data and achieve fast performance; and (4) how does one explore and interact with big data.

2.1. Exploration of Multidimensional Datasets

Large multidimensional datasets are ubiquitous. Many are behavioral data generated by people or machines, and they have a natural temporal ordering (streaming data) [7,8]. For example, network logs and financial transactions are large datasets, and they are generated by order in time. When faced with such multidimensional datasets, analysts may not know what they are looking for; they may know something is interesting only after they find it. In this setting, analysts have no idea how to formulate their queries. Thus, interactive exploration of large time series and multidimensional datasets is a key ingredient for knowledge discovery [9].

The interactive exploration system for knowledge discovery is a novel requirement in the era of "big data". The data exploration becomes a first challenge for sensemaking of large multidimensional datasets. In this paper, we focus on this problem and provide a solution for interactively exploring such datasets.

2.2. Big Data Management

Big data management methods vary based on the dataset size. If a dataset can be loaded into computer memory, the in-memory database technique should be investigated. When a dataset is larger than computer memory capacity, data prefetching and out-of-core techniques are considered [10]. If a dataset is bigger than computer disk capacity, then distributed file systems (e.g., Hadoop Distributed File System [11]) should be considered.

In this paper, our target is visual exploration of big data on an off-the-shelf desktop computer. Additionally, we consider the scenario for which the dataset size is less than computer memory. Previous research (e.g., imMens [5]) used one million or more data items as a threshold for the definition of big data. Therefore, we follow this definition and give an upper bound in our paper: the capacity of computer memory.

There are two ways to manage multidimensional (tabular) datasets: row-oriented and column-oriented formats. Abadi et al. [12] discuss the differences between them and point out that the column format can apply some compression techniques, which significantly reduce the data size and improve query performance. For example, Tableau's TDE is designed as a column-oriented format for managing big data. However, the row-oriented format is better for analyzing small transactions to "find a needle in a haystack" and is designed for an in-memory database system.

Data modeling, sampling, aggregation and indexing [5] are also used to handle big data. These methods reduce big data into smaller data. For example, Lins et al. [13] propose a new data structure called nanocube to aggregate large spatiotemporal datasets for fast querying and visualization. However, due to this, data explorations are constrained by its data structure, and it cannot directly provide flexibility and low level data details (Nanocube does not allow one to query down to any individual record). Our work emphasizes visual exploration of data fine-grained details by complex filtering and highlighting. We manage large multidimensional datasets in a row-oriented manner and utilize parallel computing to support visual exploration of big data on demand.

2.3. GPU Acceleration

Compared with the CPU, the Graphics Processing Unit (GPU) has the two following unique features for handing big data.

(1) More cores and fine levels of parallelism: The GPU has a many-core architecture, which may include thousands of cores. This feature makes the GPU specialized at compute-intensive, highly parallel computation.
(2) Higher memory bandwidth: The GPU can access data at a higher speed (usually more than 100 GB/s) or more data in a fixed time period than the CPU does.

The GPU is popular in scientific computation and visualization [14]. However, several factors have hampered its use for general-purpose computation in information visualization and visual analysis fields. Primarily, traditional algorithms in those fields are sequential, and they are not designed based on the GPU parallel architecture. However, the development of GPU libraries, such as Thrust (a parallel algorithm library) [15] and cuBLAS (GPU-accelerated basic linear algebra subprograms) [16], helps translate traditional algorithms on the GPU. Another concern is the slow data transfer from CPU to GPU memory and limited GPU memory capacity. However, transfer speeds have improved significantly thanks to PCI (Peripheral Component Interconnect) bus improvement, and now, the GPU memory capacity is quite good. All of these GPU developments make it a possible solution for handling big data to support exploratory data analysis [17].

GPU processing has deep roots in graphics and visualization, and it is good at handling millions of pixels. Researchers have proposed different methods for large data visualization based on the GPU architecture. McDonnel et al. [18] present a model to refine the traditional information visualization pipeline based on the GPU shaders. Their model emphasizes the image-space operations, addresses the mismatch between the data primitives (trees, graphs, etc.) and graphics hardware. MapD (a GPU database technology company) [19] gives a commercial product targeting large data analysis by leveraging GPU resources. The key idea of MapD is that SQL queries are compiled to native GPU code via the LLVM compiler framework (a compiler infrastructure project, which involves a collection of modular and reusable compiler and toolchain technologies) [20] to run queries in parallel. Compared with these existing work, we give a new way to utilize GPU resources to support exploratory visual

analysis of large multidimensional datasets. Specifically, we propose AVIST, which is implemented based on a GPU-centric design to explore large time series and multidimensional datasets.

2.4. Big Data Exploration

Querying and filtering are fundamental interaction techniques for data exploration. Polaris [21] features a visual querying language that integrates analysis and visualization of multidimensional datasets. Analysts can construct sophisticated visualizations by simple drag-and-drop operations. However, Polaris is incapable of managing big data. Tableau, the successor of Polaris, makes progress on big data visualization, but lacks flexible visual filtering to investigate fine data details.

Animation is one type of time multiplexing technique [22] to visualize large data. Moreover, it can effectively reveal temporal patterns by significantly improving graphical perceptions [23]. Our work highlights animation-based interactions for slicing big data into small data. Cross-filtering [1] is a method that supports flexibly drilling-down into fine-grained relationships in multidimensional datasets. Hence, we combine animation and cross-filtering interactions to achieve fast, flexible and detailed data exploration.

3. The GPU-Centric Design

3.1. Design Principles

Our design follows two important principles: (1) leveraging the GPU parallel architecture to improve performance; (2) affording flexible visual querying and filtering to support exploring fine level data details. Based on these, we carefully consider the data management and computation on the GPU:

- Data management: We store raw data in the GPU memory. In addition, all derived data are stored in the GPU memory to take advantage of its high memory bandwidth.
- Data computation: We highlight cross-filtering for slicing data from different attributes (e.g., animation for the time domain). To support such data aggregation and visualization on demand, a data dependency graph is proposed to characterize data transformations on the GPU.

3.2. Design

Our GPU-centric design has two key aspects: (1) raw data are stored on the GPU; and (2) data processing is on the GPU.

We emphasize that raw data are important for exploratory visual analysis, since they can potentially help analysts analyze each data record without any information loss (finding a needle in a haystack). The raw data are stored in the GPU memory, which enables: (1) fast data accessing based on its high bandwidth; and (2) avoiding data transfers between the CPU and GPU. Besides the raw data, derived data are also stored in the GPU memory (visual primitives are stored in GPU vertex buffer objects).

We also highlight that data aggregation and visualization are based on the GPU parallel computation. The benefits include: (1) parallel data processing based on the GPU fine level parallelism; and (2) data aggregation and visualization on demand, rather than precomputation.

Figure 1 shows the GPU-centric design. User interactions are generated on the CPU side, then they are transferred to the GPU, which triggers the GPU processing and rendering. The GPU processing is also separated into two parts: (1) general data aggregation and filtering; and (2) computation of visual primitives for each data view. In all, the GPU-centric design emphasizes that both data management and computation are handled by the GPU to gain performance.

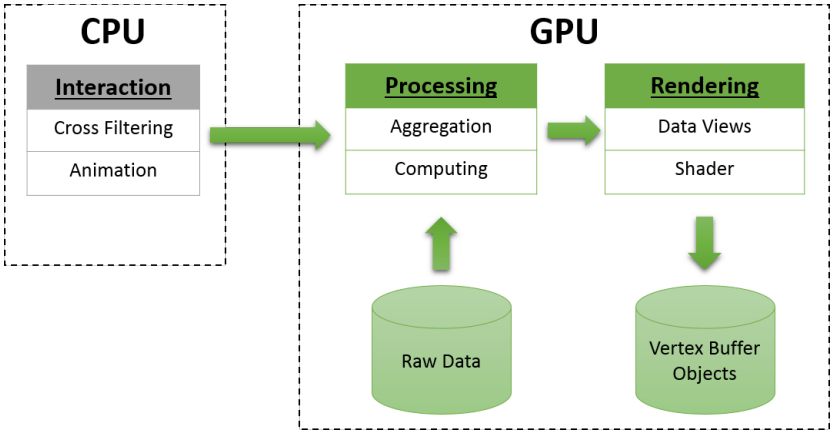

Figure 1. The GPU-centric design: user interactions are transferred from CPU to GPU. Data storage and computation are on the GPU.

3.3. Data Management

As mentioned earlier, the GPU memory capacity is one limit in our solution. To feed more data into the GPU memory, we apply a lossless compression technique to preprocess datasets.

Consider a multi-dimensional dataset, which includes different types of data. First, we identify each data type and separate them into time data, quantitative data and categorical-ordinal data. Second, we apply different compression methods considering different data types. For the categorical-ordinal data, we count all possible values and map each value into a unique ID, this key-value map is stored in main memory as metadata. For time and quantitative data, we calculate its minimum and maximum values as metadata.

On the GPU side, we organize raw data in the row-oriented format, where all data records are ordered based on their time value (we emphasize data temporal and spatial locality for fast data retrieving). We store data binary code instead of its ASCII code to save memory space. Thus, the time data occupies eight bytes, and the quantitative data needs four bytes. We map categorical-ordinal data into IDs based on their metadata. The memory requirement varies based on the possible values (one byte is required if the number of IDs is less than 256; two bytes are considered if its number is less than 65,536; other cases use four bytes). Table 1 summarizes the compression methods for preprocessing datasets.

Table 1. Data preprocessing.

Data Type	GPU Memory Binary Format	Main Memory Metadata
Time	time_t 8 bytes	minimum value maximum value
Quantitative	Int or Float 4 bytes	minimum value maximum value
Categorical-Ordinal	1∼4 bytes	Dictionary (IDs and data values)

Compared with the row-oriented format, column-oriented data organization has better performance on some analytical workloads. However, we use the row-oriented format instead of column-oriented format considering four factors: (1) exploratory analysis requires more fine-grained record details than high level aggregations, and we want to guide analysts to each data record based on their querying and filtering, rather than show them high level aggregations. (2) The column-oriented format primarily works on columns, which are treated individually. Thus, queries for each column

work efficiently, while cross-column queries need to retrieve multiple columns and to assemble that data in a complex way. The computation may be time consuming. (3) Row-oriented data have a spatial locality based on their IDs. If the data have timestamps, temporal and spatial locality can be combined together for fast data retrieval. (4) The computation of row-oriented data can be easily parallelized by leveraging GPU fine parallelism to improve performance.

3.4. Data Computation

We propose a data dependency graph to characterize data transformations on the GPU. Figure 2 shows the detailed data flow design, which is a directed acyclic graph. In the figure, rectangles represent data, while ovals are parallel computations. The graph is separated into three parts as below.

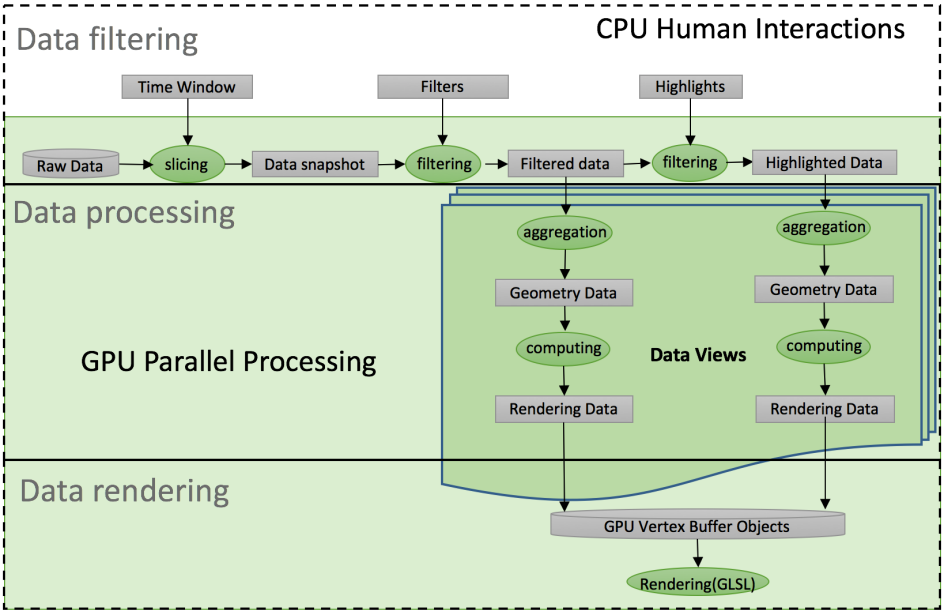

Figure 2. The data dependency graph: rectangles represent data, and ovals are parallel computations. Data filters are generated on the CPU and transferred to the GPU. Data processing and rendering are done on the GPU.

- Data filtering: Data filters (e.g., time window, filters and highlights) are generated by user interactions on the CPU side, then they are passed to the GPU side. The data flow is described as follows: (1) the time window is applied to slice raw data into data snapshots; (2) filters are applied to data snapshots by removing data records for which the user is uninterested; (3) highlighted filters emphasize important data items from the filtered data list.
- Data processing: This step transforms filtered data into visual primitives. Additionally, data transformation methods depend on each data view. We summarize data processing into two stages in each data view: (1) data aggregation, which generates geometry data (e.g., binning data for a histogram view) from filtered dataset; and (2) data computation, which transforms geometry data into visual primitives (e.g., the bar height for a histogram view).
- Data rendering: All generated visual primitives are stored in the GPU vertex buffer objects. We use GLSL (the OpenGL Shading Language) to generate the ultimate visual results, then render the visual primitives on the screen (e.g., splitting one vertex into four for rendering rectangles using the geometry shader, filling colors in triangles using the fragment shader).

The data dependency graph has several advantages. (1) It follows the cross-filtering design pattern [2], where the cross-filtering and coordinated multiple views are coupled together; (2) All data processing and visual rendering can be easily parallelized, and data aggregation is on

the fly; (3) Incremental computation can be applied to exploit temporal and spatial locality. When users play animations, frame-to-frame coherence can be exploited. In such cases, only incremental data should be considered. Moreover, user interactions are incremental and interactive, so previous queried results can be used by the next query; (4) The data dependency graph is flexible and extensible. More data filters and data views can be easily extended.

4. AVIST

We implemented AVIST based on the GPU-centric design. AVIST is written in C++. Its computation codes are based on CUDA 7.5, and visualization codes are based on OpenGL 4.5 and GLSL 4.5. The interface is coded with wxWidgets on the CPU side. We use the Thrust library [15] to accelerate sorting, scanning and reduction operations.

We describe our implementation from four aspects: (1) system architecture, which follows the Model-View-Controller (MVC) pattern [24] and separates the AVIST system into three parts: data transformations (model), data views and data filters (controller); (2) data transformation, which features the GPU parallel computation to achieve fast performance; (3) coordinated multiple views, which provide different visual aspects of multidimensional datasets; (4) user interactions, which emphasize animation and cross-filtering for slicing big data into small data.

4.1. MVC Architecture

Figure 3 shows the system architecture. Users interact with one data view, then the controller updates filters, which then triggers the data models. After all data models have been updated, the controller notifies all data views to refresh visual primitives.

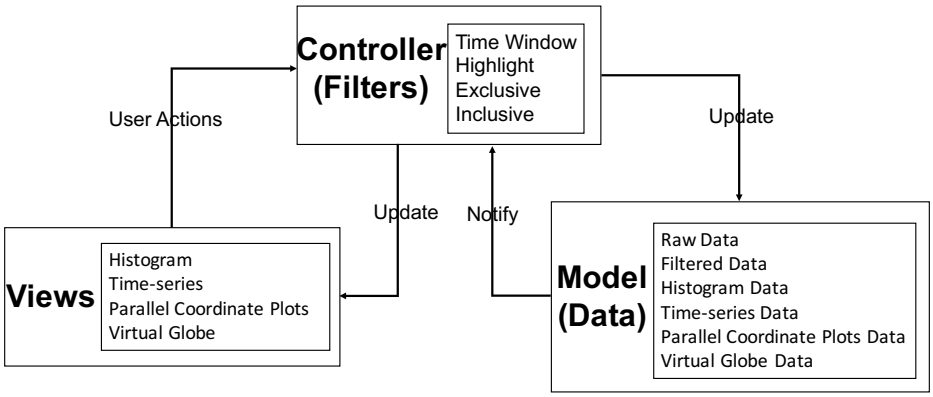

Figure 3. The Model-View-Controller (MVC) architecture in Animated VISualization Tool (AVIST). Users interact with data views to change data filters for updating data models. The controller updates the data views after it obtains the data model's notification.

The MVC architecture has three benefits. (1) The system separates filters, data and views, which gives AVIST flexibility to extend more filters, data transformations and data views in the future; (2) The model is implemented on the GPU, which highlights the GPU-centric design; (3) The coordinated multiple views can be easily applied. User interactions of one data view can update filters, which trigger data transformations and refresh other data views.

4.2. Data Transformation

AVIST highlights the parallel data transformation, which follows our data dependency graph design. The data transformations are separated into three stages: data filtering, processing and rendering.

4.2.1. Data Filtering

This stage describes data transformation from raw data into filtered data, as shown in Figure 4. First, a time window slices raw data into data snapshots, then data filters are applied to data snapshots to remove uninteresting records or highlight important ones. Because raw data are organized in time order, so data snapshots can be easily sliced (we use binary search for identifying the start and end records from raw data based on a given time window setting, rather than checking each data record in parallel). The sliced data records in data snapshots are checked by data filters in parallel, and then, they are reduced to a filtered dataset.

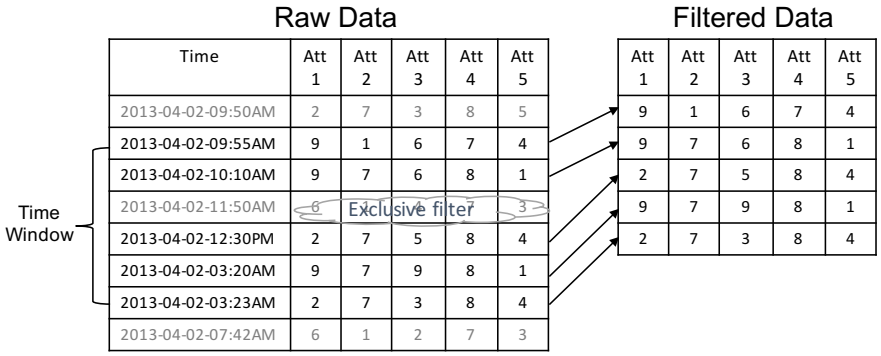

Figure 4. Data transformations from raw data to a filtered dataset.

4.2.2. Data Processing

Data processing is view dependent. We consider the implementation of four data views: histogram view, time series view, parallel coordinate plots and virtual global view (more details about them can be found in Section 4.3).

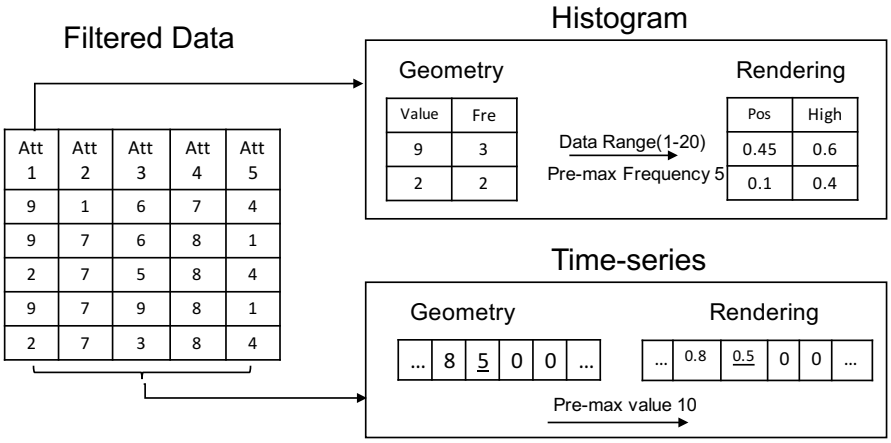

Figure 5. Data transformations from filtered data into visual primitives.

Figure 5 demonstrates the data transformation of histogram and time series views. Based on the user-selected column in the histogram view, AVIST retrieves the corresponding data, aggregates and then visualizes them. The steps are as follows:

1. Sort one data column to get unique values and their frequency.
2. Get the X position of each unique value based on the data range (X_{pos} is the position column of the rendering table in the histogram box in Figure 5). This value is calculated as:

$$X_{pos} = (Column_{Value} - Column_{minimum}) / range_{column}$$

3. Get the Y position of each unique value based on previous maximum frequency, and then, update this value for the next computation (Y_{pos} is the height column of the rendering table in the histogram box in Figure 5). They are calculated as:

$$Y_{pos} = Column_{Freq}/Value_{PreMaxFreq}, Value_{PreMaxFreq} = Max\left(Value_{PreMaxFreq}, Column_{Freq}\right)$$

The previous maximum frequency is the maximum of all previous frequency values in histogram geometry data. If users create a new histogram view by choosing a different column, a new previous maximum frequency value of that column is calculated.

Data transformation of the time series view is straightforward. The GPU counts the number of data records in a filtered dataset and transforms this to a height value (based on the previous maximum value).

Data transformations of Parallel Coordinate Plots (PCPs) are more complex than other views. Figure 6 shows the data transformations. First, the GPU generates a compact list based on chosen axes. Then, the data flow is separated into two parts, including the vertices and edges generations.

The steps for vertex generation are described as follows:

1. Sort each column individually to remove duplicated values.
2. Assemble the compact columns into three arrays: vertex array, size array and offset array.
3. Generate each node position: the X-axis is based on the selected order of each column, and the Y-axis is based on the item value and its dimension range.

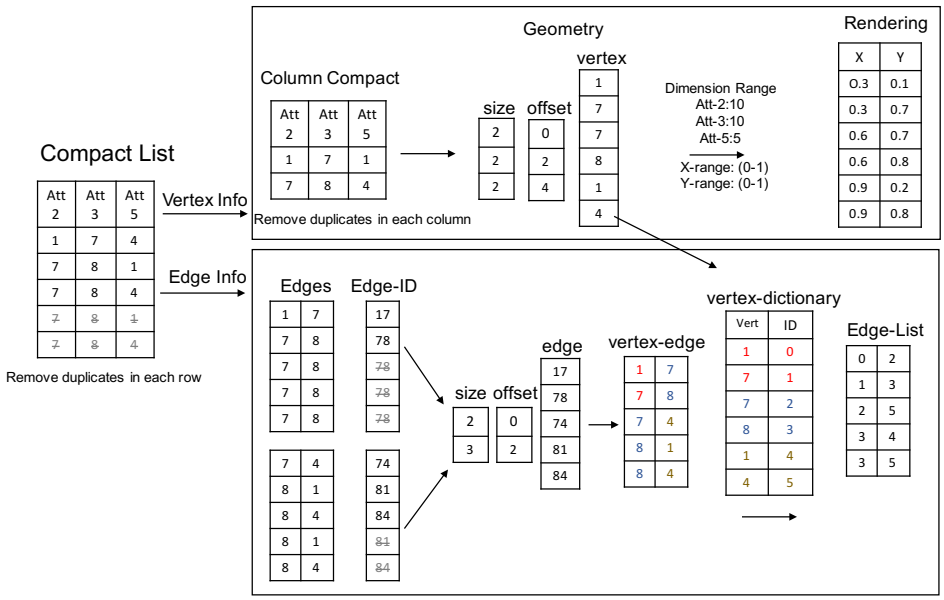

Figure 6. Data transformations of parallel coordinate plots, which are separated into three parts: compact list, vertex and edge generations.

The steps for generating an edge list are as follows:

1. Two neighboring columns are grouped into one array (Edge-ID array) as edges. The value in this array is calculated as: $Edge\text{-}ID = value_{columnA} \times range_{columnA} + value_{columnB}$
2. Sort the Edge-ID array and remove duplicates.
3. Compact multiple Edge-ID arrays into three data arrays: edge array, size array and offset array.
4. Convert the edge array into the vertex-edge array.
5. Replace each vertex value in the vertex-edge array with its order based on the vertex-dictionary to generate Edge-List array.

The virtual global view is a special case of parallel coordinate plots with two axes. The difference is that the vertex position is 3D in the virtual global view, and there are extra steps to generate 3D positions based on longitudes and latitudes. In the implementation, we use Bezier curves to link two geolocations for revealing their relationships.

4.2.3. Data Rendering

AVIST stores visual primitives into GPU vertex buffer objects (VBOs), which offer substantial performance gains and avoid data transfers between the CPU and GPU. However, visual primitives in the VBOs are vertexes and edges. In order to generate rectangles, areas or other advanced shapes, we use GLSL for post-rendering. For example, we split one vertex into four vertices to render rectangles in the histogram view. We transform vertices into line strips to generate areas in the time series view.

4.3. Coordinated Multiple Views

Four data views are provided in AVIST, which are histogram view, time series view, parallel coordinate plots and virtual global view. The data views work together with data filters, which enables cross-filtering of multidimensional datasets. Data views also support direct visual filtering with brushing interactions (e.g., an analyst first selects highlighted filters in solo mode, then he or she brushes one bar in the histogram view to highlight corresponding data records).

- A histogram view shows the data distribution of a sliced data snapshot. Analysts can choose different dimensions to explore aggregated information.
- A time series view shows data aggregation about certain filtered events over a period of time. When an analyst changes data filters, the time series view clears previous visualization and redraws everything.
- Parallel coordinate plots show data details. Analysts can select multiple data dimensions to generate their customized parallel coordinate plots. The axes are organized based on their selected order.
- A virtual global view shows geographical information. The Bezier curves are used to link locations on the virtual globe for representing data relationship.

4.4. User Interactions

Figure 7 shows the control panel of AVIST, which provides animation and filtering interactions.

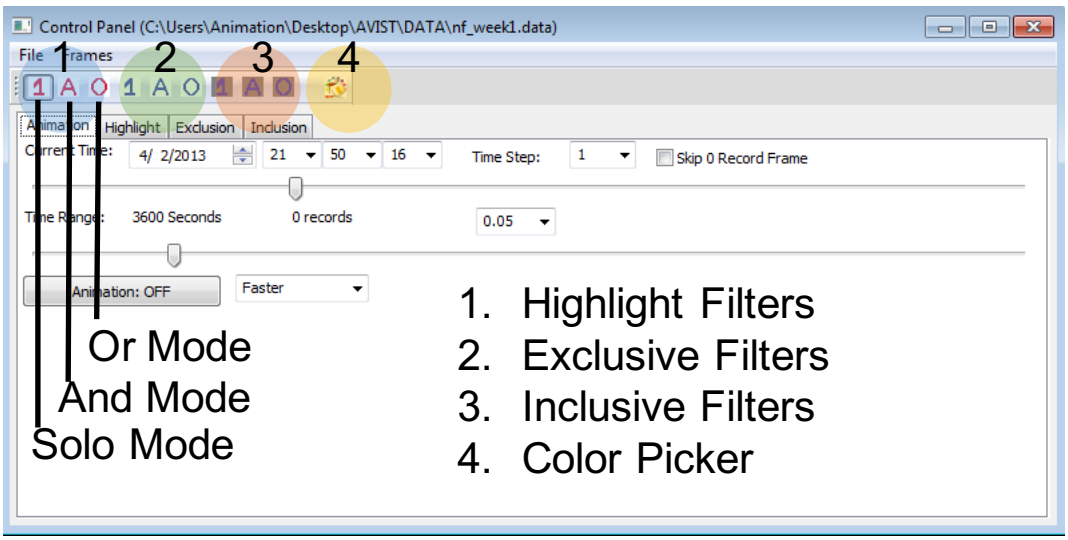

Figure 7. The control panel of AVIST. Three filters (highlight, exclusive and inclusive) with three modes (solo, and, and or) are provided.

4.4.1. Animation

The animation interactions include automatic forward playback, dragging a time window bar and adjusting animation speed. By updating the current time and time range, the time window is customized to slice data into snapshots, which are analyzed and visualized in correlated data views. By combining automated animation with these correlated views, AVIST can provide temporal changes in the datasets.

4.4.2. Filtering

Three different filters are implemented in AVIST.

- Highlight filters: making selected data items stand out from the rest with different colors.
- Exclusive filters: removing uninteresting data items.
- Inclusive filters: the exact opposite of exclusive filters, removing all data items except those marked by an analyst.

Each data filter has three modes:

- Solo mode: allowing only one filtered item in the current filter set.
- And mode: combing several filters together as a whole and selecting data records to satisfy all conditions.
- Or mode: selecting data records to meet at least one of all of the filter conditions.

With these three filters and their modes, analysts can generate complex nested data filters and apply them to different data views, which can help them drill down and "find a needle in a haystack" on demand.

4.4.3. Capacities

The animation interactions can slice big data into fine-grained subsets by narrowing down the size of the time windows. Filtering interactions can extract relevant records for analysts by focusing on small data subsets. The cross-filtering can be achieved by combing different filters and multiple data views. These interactions transform large data into small data by removing uninteresting items or highlighting important ones, which provide the flexibility of exploring large multidimensional datasets.

5. Performance

We evaluate AVIST performance on a desktop computer, running Windows 7 Enterprise, which is equipped with an Intel i7 processor and NVIDIA GeForce GTX 680 graphics card with 4 GB memory. We use a network traffic dataset from the first case study as our benchmark.

Figure 8 shows the performance about data filtering and transformation. It describes a linear relationship between performance and the number of queried data records, except for the time series data. The reason is that the aggregated information of the time series view does not require much computation. The data processing to generate parallel coordinate data is the most time consuming part in AVIST. To further investigate detailed GPU performance in parallel coordinate plots (eight axes), we characterize data transformations into three stages, including: (1) compact list generation; (2) vertex generation; and (3) edge generation. The performance is shown in Figure 9, where the data generation of the compact list needs more time than others. The reason is the compact list is generated by the raw data, where it removes duplicated rows. The vertices and edges generations are based on the compact list, and their data records highly reduced by the compact list. The number of axes also has an impact on performance. Figure 10 shows the performance of parallel coordinate data with 2, 4 and 8 axes. We see that the time decreases with increasing axes. However, the number of columns has a limited impact.

We have implemented a CPU version for parallel coordinate plots. The comparison between the CPU and GPU is shown in Figure 11. We see that the GPU version is faster than the CPU version.

The reason is that the performance is heavily based on removing duplicates. We have different solutions to achieve this on the CPU and GPU platforms. In the CPU version, we use the map and set containers from the C++ Standard Template Library (STL) to remove the duplicated data items. Their implementation is based on a red-black tree [25]. Thus, the time complexity of inserting N items into a set container to remove duplicates is $O(Nlog(N))$. In the GPU version, we use the sort operation from the Thrust library to remove duplicates. Its implementation is based on the radix sort algorithm [26]. The complexity of it is $O(NK)$, where K is the number of bits in one element (the value of K is 64 in our implementation).

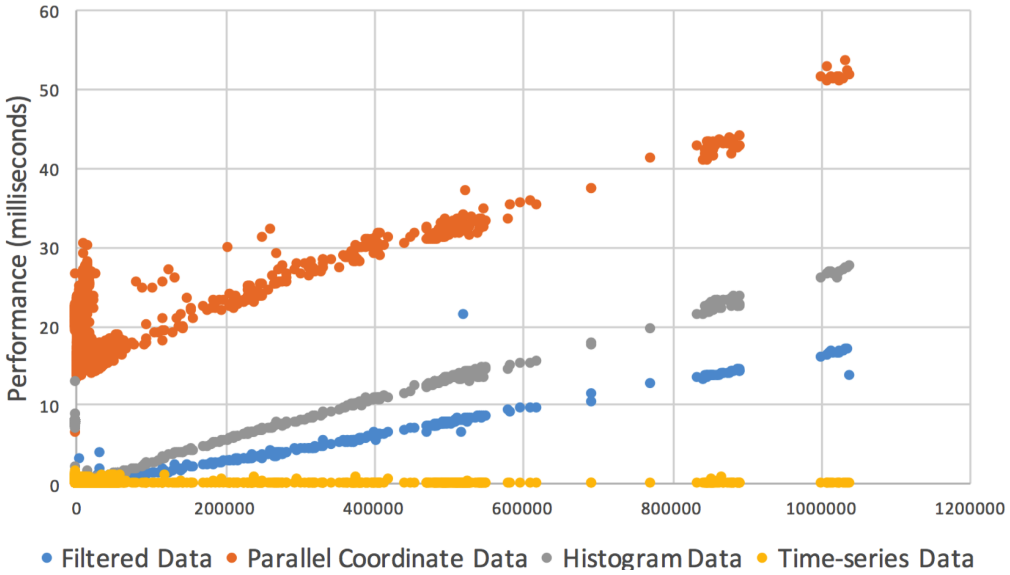

Figure 8. The performance of AVIST. The X-axis is the number of queried data records, and the Y-axis is the time (milliseconds). This scatter plot shows the performance of four kinds of data: filtered data, parallel coordinate data (eight axes), histogram data and time series data.

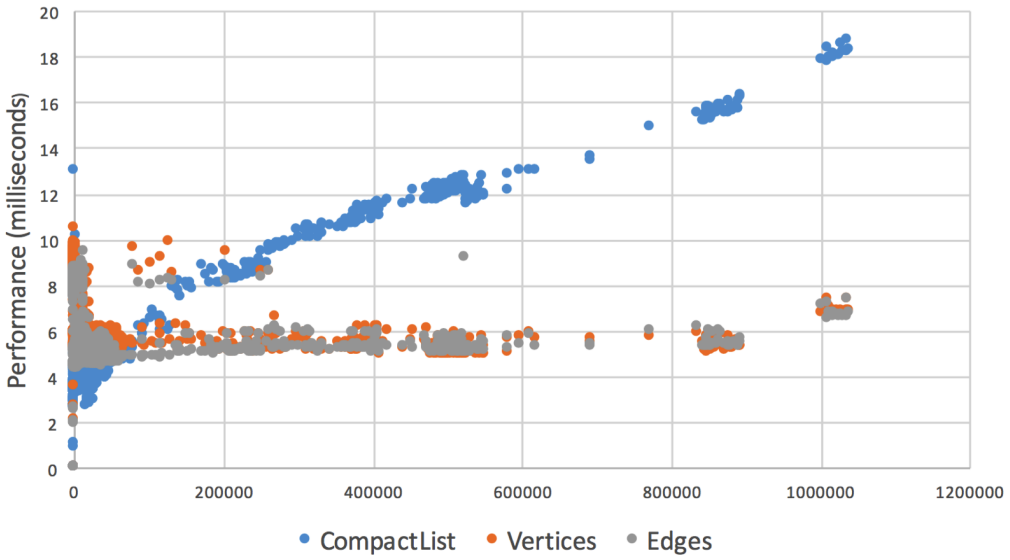

Figure 9. The detailed performance for generating parallel coordinate data. The X-axis is the number of queried data records, and the Y-axis is the time (milliseconds). Three steps are characterized in the plot, including the data generations of compact list, vertices and edges.

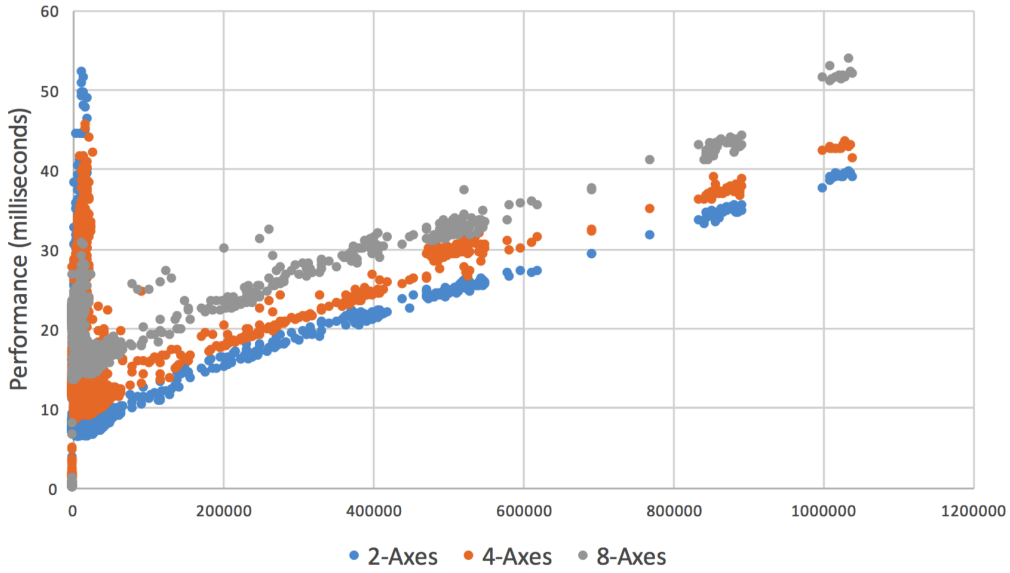

Figure 10. The performance for generating parallel coordinate data with different numbers of axes. The *X*-axis is the number of queried data records, and the *Y*-axis is the performance (milliseconds). The figure includes three cases: 2 axes, 4 axes and 8 axes.

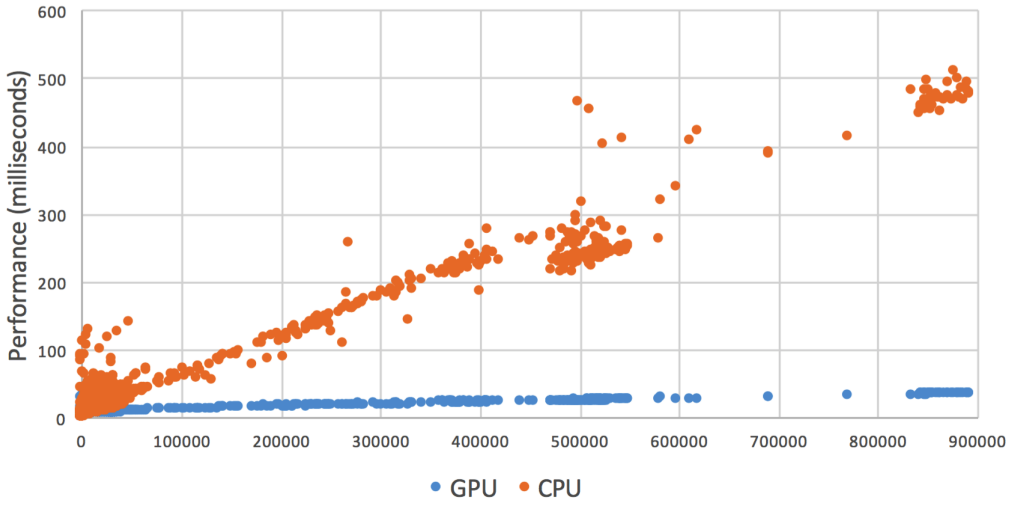

Figure 11. The performance comparison between the CPU and GPU in parallel coordinate plots (two axes). The *X*-axis is the number of queried data records, and the *Y*-axis is the time (milliseconds).

6. Usage Scenario

In this section, we use two usage scenarios to demonstrate how AVIST can support visual exploration of big data and help analysts identify hidden patterns, infer and verify new hypotheses.

6.1. Network Flow Analysis

Exploratory visual analysis of network logs is critical for detecting potential cyber threats and network intrusions. Visual Analytics Science and Technology (VAST) 2013 Mini Challenge 3 targets the analysis of Big Marketing company networks, and it provides big network traffic logs [27]. One dataset is one week worth of network flow, which includes 46,138,310 records and 16 dimensions.

Suppose that Alice is an intelligence analyst. She is assigned to identify network threats for Big Marketing company. At first, she compresses the dataset into binary format and loads it to AVIST. Figure 12 shows key steps of Alice's analytical process.

Data exploration from overview to details: At the beginning, Alice specifies the time window size (120 s in this example), then plays the animation. The overview of network traffic is visualized in the time series view, shown in Subgraph 1. Alice finds an unusual behavior that the network is crashed from 2 April 2013 9:40 a.m. to 3 April 2013 3:26 a.m. She wants to investigate the details before the network crash, so she chooses data dimensions in the order of *firstSeenSrcIp*, *firstSeenSrcPort*, *firstSeenDestIp* and *firstSeenDestPort* to generate parallel coordinate plots in Subgraph 2. Then, she plays the animation again and finds that four source IPs 10.10.6.2, 10.6.6.6, 10.7.6.5, 10.7.7.10 scan the destination IP 172.30.0.4 during 2 April 2013 5:20 a.m. in Subgraph 3.

Flexible filtering to reveal hidden patterns: Subgraph 4 shows the network traffic distribution of *firstSeenDestPort*. Alice realizes that most of the network records are related with port 80, and she hypothesizes that these are normal transactions. She removes these data records with port 80 (Subgraph 5) to investigate buried patterns. Then, she plays the animation again and investigates port scan activity, shown in Subgraph 7. She captures this behavior by highlight filters as shown in Subgraph 6.

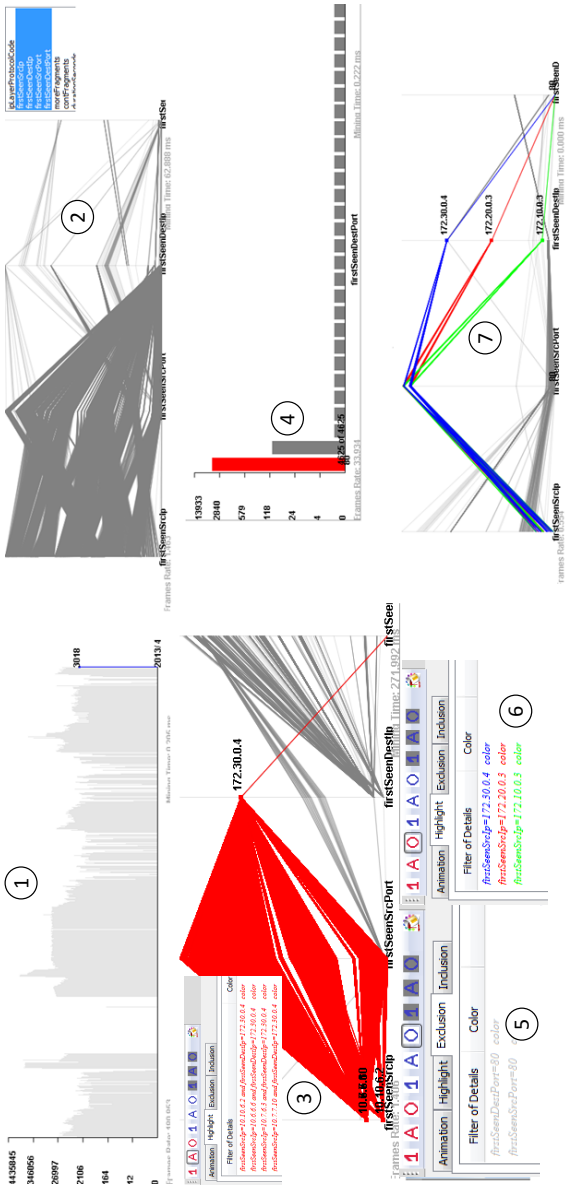

Figure 12. A usage scenario where AVIST helps an analyst visually explore large network logs. Seven steps are presented in this figure, and Section 6.1 gives detailed explanations.

In this scenario, Alice iteratively plays animations and tries different filters to identify potential network threats. She can easily constrain the time and data attributes to capture subtle and hidden relations. For example, Alice investigates port scan activity by removing network flow traffic related with destination port 80 in Subgraph 7. However, she cannot identify this activity from Subgraph 2 in the first round animation.

6.2. International Trade Analysis

Making sense of international trade is important for governments and policy makers. With the world economic growth, world trade transactions have become large and complex. In this case study, we obtain big international trade transactions from PIERS Import/Export trading system [28], which provides portfolio analysis for U.S. waterborne trade activity. The dataset includes 10,735,092 records and 10 dimensions about 2013 U.S. waterborne imports. Within 10 dimensions, each data record features a U.S. port code and a foreign port code, which illustrates each trade geospatial relationship.

Suppose that Mike is an economic analyst, and he wants to gain insights from the international trade activities. Mike decides to explore this dataset using AVIST. Figure 13 shows Mike's visual analytical results.

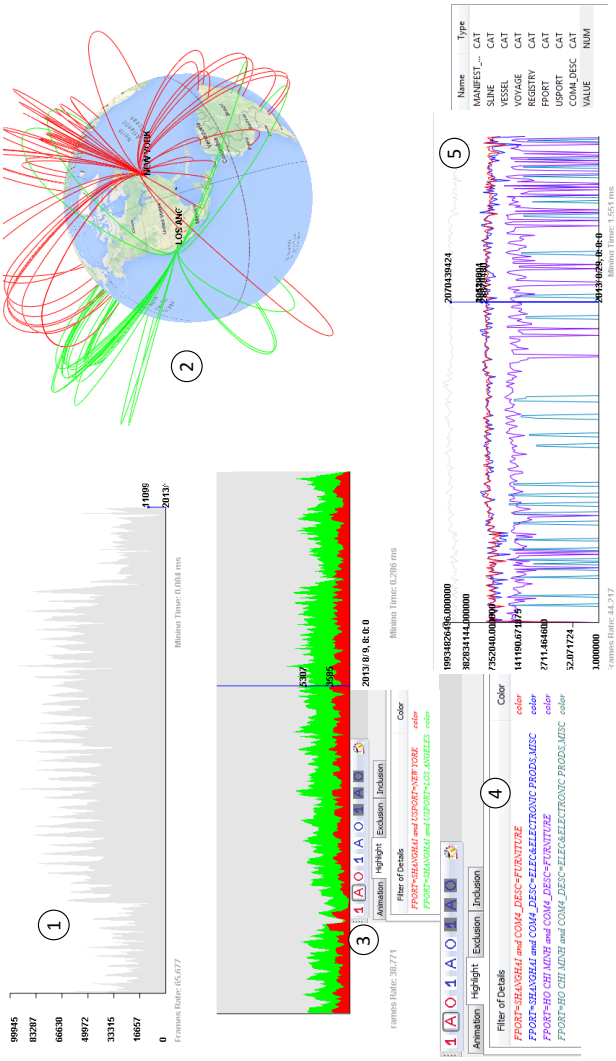

Figure 13. A usage scenario where an analyst uses AVIST to visually explore big international trade transactions. The detailed discussions are presented in Section 6.2.

Overview exploration: Mike specifies the time window (86,400 s or one day) before playing the animation. The time series view shows the overview of the world trades in Subgraph 1, which presents that trade activities are clearly separated into four quarters. The second and third quarters have more transactions than others.

Hypotheses generation and verification: Mike hypothesizes that the international trades may have some spatial patterns. International traders prefer neighboring countries. To verify this, Mike highlights two U.S. ports: Los Angeles (LA) and New York (NY), then he plays the animation. Subgraph 2 shows one snapshot in virtual globe view, which indicates that LA has more trades with Pacific countries, while NY is strongly related with Western Europe. Mike narrows down the data records and he focuses on the trades from Shanghai (SH) to LA and NY. Then, he plays the animation to refresh the time series view in Subgraph 3. He finds that the trades from SH to LA are roughly twice of these of SH–NY, which supports his hypotheses.

Making sense of big data: The international trades can characterize each country's economy. Mike generates four filters to compare the economies of China and Vietnam. He chooses Shanghai port and Ho Chi Minh port to represent the two countries. He emphasizes the trades related to furniture and electronics, as shown in Subgraph 4. He is more interested in the money value rather than the number of trades, so he clicks the value in the list box and plays the animation. Subgraph 5 is the time series view with four highlighted line charts, which shows that the two kinds of trades are more balanced in China, while Vietnam has more furniture exports than electronics exports.

7. Lessons Learned

In this section, we summarize lessons learned for designing and implementing AVIST. We begin our discussion with the trade-offs of selecting different techniques for designing big data visual analysis tools.

- Aggregated visualization vs. atomic visualization: These are two strategies to present big data visually. Atomic visualization uses one visual element for each data record [29]. Pixel-oriented visualization is an extreme case, where each data record is mapped to a single pixel, its color indicating some attribute of this record [30]. However, this technique is constrained by display resolution, which sometimes leads to the over-plotting problem [5]. Different from atomic visualization, aggregated visualization highlights one visual primitive for multiple data records. It allows users to see a subset of data as coarse views with interactions (e.g., zooming and animation). In the case of limited screen pixels, aggregated visualization is more useful than atomic visualization because it can avoid the over-plotting problem. For instance, in AVIST, after a user clicks on a bar in the histogram view, the parallel coordinate plots highlight thousands of lines to reveal corresponding details. Thus, in a setting of limited screen pixels, aggregated visualization is more scalable than atomic visualization. Moreover, aggregated visualization can help reduce the performance overhead since it does not plot all data at once.
- Pre-aggregation vs. aggregation on demand: Pre-aggregation is a strategy to achieve high level data exploration by reducing big data into small data. imMens [5] is a pre-aggregation big data visual querying system. It aggregates data using the data cube technique, then it allows users to explore the aggregated data. This method is constrained by: (1) preprocessing methods; (2) huge memory requirements (derived data may be larger than original data); and (3) lacking flexible filtering and querying. In contrast, aggregation on demand has no such constraints, and it emphasizes aggregated data generated on the fly. Thus, it supports flexible data filtering. AVIST aggregates data based on users' demands, and features the GPU-centric design to enable online data aggregation.
- Column-oriented vs. row-oriented: These are two basic data management methods, and both of them have pros and cons. In the "big data" era, column-based methods gain more attentions, since a columnar database has two features. First, it has a better compression ratio by storing similar things together, and it reduces IO (Input/Output) cost during data transferring from

disk to memory. Second, it supports high level analytical workloads well. On the other hand, row-based databases are better for On-Line Transaction Processing (OLTP) applications, which support frequently reading and writing small transactions. The key advantage of the row-oriented data format is efficiently retrieving small data to find a "needle in a haystack". Thus, we use the row-oriented format for querying fine data details.

- Analytically-driven vs. exploration-driven: These are two approaches for sensemaking of big data. Analytically-driven systems focus on "known-unknowns" insight synthesis, and they care about integrating statistics and machine learning techniques with human interactions and visualizations. However, exploration-driven applications emphasize "unknown-unknowns" discovery and highlight data exploration and interaction techniques. Thus, if users do not know what they want to know from big data, exploration-oriented applications are better choices. Users can identify some unexpected knowledge by exploring the data.
- Scaling out vs. scaling up: These are two ways to improve performance. The scaling out techniques are general approaches to handle big data, where more computers are added based on demand. Distributed computing technology is the key to manage many computers. However, distributed computing is designed for long batch jobs. Nowadays, researchers in this area concentrate on interactive analysis tasks and propose new frameworks (e.g., Spark [31]). Even though, most of them are analytically driven, which emphasize scaling data mining and machine learning from small data into big data, rather than exploring large datasets. As an alternative, the scaling up techniques add more resources on a single machine to boost performance. In this paper, we emphasize performance improvement by fully exploiting GPU resources. Thus, we achieve visual analysis of big data in real-time performance on an off-the-shelf computer.

Through the designing of AVIST, we have shown GPU-based data transformations. Lessons learned from designing GPU-based visual exploration systems are summarized as follows.

- Data management on the GPU: The GPUs are compute-intensive processors. They are designed to perform data-parallel computation, in which a single instruction works over a large block of data. Working in blocks of data is more efficient than working with a single cell at a time. A large block of parallel working units means high throughput computation. This kind of throughput-oriented computation needs high memory bandwidth. Thus, the GPU favors array-based data structures.

  When programming on the CPU, designers can write to any location in memory at any point in their program. However, when programming on the GPU, designers access the GPU memory in a much more structured manner. The GPU is a SIMD (Single-Instruction Multiple-Data) architecture, which ensures that the computation on one data element cannot affect another. The only values that can be used in a GPU kernel are kernel input parameters and the GPU global memory reads. In addition, the output of a GPU kernel should be independent. GPU kernels cannot have random writes into the GPU global memory; each thread of a GPU kernel may perform writes to a single element in the output memory.

  Considering GPU memory organization, designers need to pay special attention to the data memory layout. Compared to the CPU, the GPU has different cache optimization. Designers need to explicitly cache heavily-used variables in GPU shared memory to improve performance. GPU shared memory is limited in terms of size, and inappropriate usage of GPU shared memory may hamper the performance. The reason is that the GPU shared memory is divided into memory banks. If two memory addresses occur in the same bank, then a bank conflict occurs [32] during which the memory access is done serially, losing the advantages of parallel access.

  Additionally, designers should take care of how to access GPU global memory efficiently. GPU global memory features combining multiple memory accesses into as few transactions as possible to minimize its bandwidth, which is referred to as memory coalescing.

Thus, designers should be aware of improper GPU memory access patterns (e.g, sparse memory access, misaligned memory access, etc.).

In AVIST, we carefully consider the GPU programming pitfalls listed above. Our data structures are based on data arrays, and we have removed duplicated data items in those data arrays to reduce computation overhead. We have compacted multiple small data arrays into one array to have a better GPU memory access pattern.

Designing an efficient GPU kernel is nontrivial, especially when considering GPU memory organization. Instead of designing complex GPU kernels, we implement data transformations based on existing GPU libraries (e.g, Thrust). In AVIST, we use the sort operation to eliminate duplicates and obtain unique values; we also use the reduction operation to organize data arrays.

- Data visualization on the GPU: We highlight the usage of the GPU VBOs to improve the performance of big data visualizations. The basic visual primitives for GPU rendering are vertices or triangles. To generate complex visual primitives (e.g., bars and areas) or obtain advanced visual features, we use the GLSL, which transforms GPU geometric primitives into a raster image. The GPU features its rendering pipeline with three programmable shader stages: vertex shader, geometry shader and fragment shader. Marroquim et al. [33] provide a detailed description of each stage. In fact, the graphics pipeline is independent across stages, which makes rendering performance significantly improved. Thus, the GPU is a powerful weapon for rendering big data.
- Data interaction on the GPU: The data interaction in AVIST emphasizes highlighting, brushing and filtering. The implementation of these interactions includes two stages.

  (1) Keyboard and mouse events are collected on the CPU side, and they are transformed from the windows 2D space into the image space based on the view port setting of the GPU. After that, these events are assembled and transferred to the GPU side.

  (2) The GPU parses these data and links the current visual primitives back to raw data. To achieve this, the GPU VBOs map visual primitives back to the GPU global memory. Then, the GPU filters each visual primitive in parallel to get the filtered ones. After that, the GPU traces back to raw data based on the filtered ones and continues data transformations for the next frame.

In summary, designing GPU-based big data visual analysis systems requires significant GPU knowledge, which may become too esoteric for researchers in the field of information visualization. Information visualization researchers may need to collaborate with experts in high performance computing and graphics to narrow the gap between big data and its insights.

8. Limitations

In this work, we exploit the powerful resources of GPUs for interactive big data visualization. Thus, the GPU memory size and computation capacity are the key limitations.

- GPU memory: The GPU memory capacity is limited, while the volume of big data can be far beyond its capability. We have applied a lossless compression technique in AVIST to shrink the data size. However, this technique cannot scale to some bigger datasets.
- GPU computation: The computational capability of GPUs has reached tera-FLOPS (Floating-point Operations Per Second), which may be ten-times faster than state-of-the-art CPUs. However, GPUs perform well for arithmetic intense computation with regular memory access patterns. Thus, they are incapable of some problems with unpredicted memory accesses and highly sequential solutions (e.g., graph traversal). GPUs are specialized for certain problems and are incapable of achieving significant performance gains for all problems.

There is much room to improve our work. We plan to have detailed performance comparisons between our work with existing tools or systems, such as Nanocubes and MapD. We also want to investigate unstructured or semi-structured data analysis, such as big texts analytics based on the GPU.

9. Conclusions

In this paper, we focus on interactive visual analysis of large multi-dimensional datasets and contribute a GPU-centric design for visual exploration of such datasets. We emphasize the GPU parallel resources to improve performance. To achieve this, we propose a data dependency graph design to characterize data computations on the GPU. As a proof-of-concept, we implemented AVIST based on the GPU-centric design. We have described its implementation details and performance analysis.

We have demonstrated how AVIST can help analysts gain insights into big data with two usage scenarios. We have summarized the lessons learned and discussed the trade-offs of different techniques for designing big data visual analysis systems. We also summarized the pitfalls and advantages for implementing GPU-based systems for interactive information visualization applications.

Acknowledgments: This research was partially supported by NSF grant IIS-1447416.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Weaver, C. Cross-Filtered views for multidimensional visual analysis. *IEEE Trans. Vis. Comput. Graph.* **2010**, *16*, 192–204.
2. Weaver, C. Multidimensional visual analysis using cross-filtered views. In Proceedings of the IEEE Symposium on Visual Analytics Science and Technology, Columbus, OH, USA, 21–23 October 2008; pp. 163–170.
3. Tableau Software. Available online: http://www.tableau.com/ (accessed on 27 September 2016).
4. Wesley, R.; Eldridge, M.; Terlecki, P.T. An analytic data engine for visualization in tableau. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Athens, Greece, 12–16 June 2011; pp. 1185–1194.
5. Liu, Z.; Jiang, B.; Heer, J. imMens: Real-time Visual Querying of Big Data. In Proceedings of the 15th Eurographics Conference on Visualization, Leipzig, Germany, 17–21 June 2013; pp. 421–430.
6. Gray, J.; Chaudhuri, S.; Bosworth, A.; Layman, A.; Reichart, D.; Venkatrao, M.; Pellow, F.; Pirahesh, H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.* **1997**, *1*, 22–53.
7. Canny, J.; Zhao, H. Big data analytics with small footprint: Squaring the cloud. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, IL, USA, 11–14 August 2013; pp. 95–103.
8. Zikopoulos, P.; Eaton, C. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Athens, Greece, 12–16 June 2011; McGraw-Hill Osborne Media: New York, NY, USA, 2011.
9. Idreos, S.; Papaemmanouil, O.; Chaudhuri, S. Overview of Data Exploration Techniques. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Melbourne, Australia, 31 May–4 June 2015; pp. 277–281.
10. Battle, L.; Chang, R.; Stonebraker, M. Dynamic Prefetching of Data Tiles for Interactive Visualization. In Proceedings of the International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 1363–1375.
11. Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. The hadoop distributed file system. In Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies, Incline Villiage, NV, USA, 3–7 May 2010; pp. 1–10.
12. Abadi, D.J.; Madden, S.R.; Hachem, N. Column-stores vs. row-stores: How different are they really? In Proceedings of the ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, 9–12 June 2008; pp. 967–980.
13. Lins, L.; Klosowski, J.T.; Scheidegger, C. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graph.* **2013**, *19*, 2456–2465.
14. Balsa, R.M.; Gobbetti, E.; Iglesias, G.J.; Makhinya, M.; Marton, F.; Pajarola, R.; Suter, S.K. State-of-the-Art in Compressed GPU-Based Direct Volume Rendering. *Comput. Graph. Forum* **2014**, *33*, 77–100.

15. Thrust - Parallel Algorithm Library. Available online: http://docs.nvidia.com/cuda/thrust/ (accessed on 27 September 2016).
16. cuBLAS - Basic Linear Algebra Subprograms on CUDA. Available online: http://docs.nvidia.com/cuda/cublas/ (accessed on 27 September 2016).
17. Pawliczek, P.; Dzwinel, W.; Yuen, D.A. Visual exploration of data by using multidimensional scaling on multicore CPU, GPU, and MPI cluster. *Concurr. Comput. Pract. Exp.* **2014**, *26*, 662–682.
18. McDonnel, B.; Elmqvist, N. Towards Utilizing GPUs in Information Visualization: A Model and Implementation of Image-Space Operations. *IEEE Trans. Vis. Comput. Graph.* **2009**, *15*, 1105–1112.
19. MapD Technology. Available online: https://www.mapd.com/ (accessed on 27 September 2016).
20. LLVM Complier Infrastructure. Available online: http://llvm.org/ (accessed on 27 September 2016).
21. Stolte, C.; Tang, D.; Hanrahan, P. Polaris: A system for query, analysis, and visualization of multidimensional databases. *Commun. ACM* **2008**, *51*, 75–84.
22. Fekete, J.D.; Plaisant, C. Interactive information visualization of a million items. In Proceedings of the IEEE Symposium on Information Visualization, Boston, MA, USA, 28–29 October 2002; pp. 117–124.
23. Heer, J.; Robertson, G. Animated transitions in statistical data graphics. *IEEE Trans. Vis. Comput. Graph.* **2007**, *13*, 1240–1247.
24. Hatanaka, I.; Hughes, S.C. Providing Multiple Views in a Model-View-Controller Architecture. U.S. Patent 5,926,177, 20 July 1999.
25. Musser, D.R.; Derge, G.J.; Saini, A. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*; Hendrickson, M., Ed.; Addison-Wesley Professional: Boston, MA, USA, 2009.
26. Merrill, D.; Grimshaw, A. High Performance and Scalable Radix Sorting: A Case Study of Implementing Dynamic Parallelism for GPU Computing. *Parallel Process. Lett.* **2011**, *21*, 245–272.
27. The Visual Analytics Science and Technology (VAST) Challenge 2013. Available online: http://vacommunity.org/VAST+Challenge+2013 (accessed on 27 September 2016).
28. PIERS Global Intelligence Solutions. Available online: https://www.ihs.com/products/piers.html (accessed on 27 September 2016).
29. Shneiderman, B. Extreme visualization: Squeezing a billion records into a million pixels. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, 9–12 June 2008; pp. 3–12.
30. Keim, D.A. Pixel-Oriented visualization techniques for exploring very large data bases. *J. Comput. Graph. Stat.* **1996**, *5*, 58–77.
31. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. In Proceedings of the USENIX Conference on Hot Topics in Cloud Computing, Boston, MA, USA, 22–25 June 2010; pp. 1–10.
32. Mark, H.; Shubhabrata, S.; John, D. Parallel prefix sum (scan) with CUDA. *GPU Gems* **2007**, *3*, 851–876.
33. Marroquim, R.; Maximo, A. Introduction to GPU Programming with GLSL. In Proceedings of the 2009 Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing, Rio de Janeiro, Brazil, 11–14 October 2009; pp. 3–16.