P5: Portable Progressive Parallel Processing Pipelines for Interactive Data Analysis and Visualization

Jianping Kelvin Li and Kwan-Liu Ma

Abstract— We present P5, a web-based visualization toolkit that combines declarative visualization grammar and GPU computing for progressive data analysis and visualization. To interactively analyze and explore big data, progressive analytics and visualization methods have recently emerged. Progressive visualizations of incrementally refining results have the advantages of allowing users to steer the analysis process and make early decisions. P5 leverages declarative grammar for specifying visualization designs and exploits GPU computing to accelerate progressive data processing and rendering. The declarative specifications can be modified during progressive processing to create different visualizations for analyzing the intermediate results. To enable user interactions for progressive data analysis, P5 utilizes the GPU to automatically aggregate and index data based on declarative interaction specifications to facilitate effective interactive visualization. We demonstrate the effectiveness and usefulness of P5 through a variety of example applications and several performance benchmark tests.

Index Terms—Information visualization, progressive analytics, visualization software, GPU computing, data exploration

1 Introduction

Visualization is useful for understanding, analyzing, and exploring complex data. Many visualization systems have been developed to help users perform reasoning, analysis, and exploration tasks by combining automated statistical computations and interactive visualizations. For supporting effective visual analysis, systems must provide interactive performance to ensure users can remain focused and maintain their attention [8, 9, 38]. As data continues to grow in volume and complexity, visualization systems need to leverage high-performance computing to accelerate data processing and rendering. Most existing visualization libraries and toolkits [7, 35, 36, 44], however, only use sequential computation for performing data transformations and visualizations; therefore, they are not capable of handling large data and the associated processing. Some of these toolkits support crafting visualizations through the use of declarative grammars [20, 45], but they do not support constructing a high-performance visualization system. As multicore CPUs and manycore GPUs have become commonly available in mainstream computers, a few toolkits [23, 33, 34] have been made to exploit parallel computing to improve the performance of visualization rendering. Despite these advances, effective visualization of big data requires not only decreasing the rendering latency but also increasing the data transformation speed. Moreover, it is difficult to provide interactive performance for visual analysis when the scale of the data size exceeds the processing capacity of the underlying visualization system.

To support interactive analysis of big data, progressive analytics has recently emerged as a popular approach for (1) delivering incremental results to maintain an acceptable level of responsiveness, and (2) allowing users to interact with the intermediate results and better control the exploration process. Several visualization systems [6, 31, 40], user studies [17, 46], and frameworks [30, 37, 42] have provided design guidelines for developing progressive data analytics. However, we lack visualization toolkits that leverage these design guidelines for building progressive visualization systems. Moreover, progressive analytics and GPU-based parallel computing can be combined to provide a better solution for interactive analysis of big data. A declarative visualization toolkit incorporating the capabilities of parallel computing and progressive analytics can make building high-performance and scalable visualization systems a much easier job.

In our prior work, we developed P4: Portable Parallel Processing Pipelines [24], a web-based visualization toolkit that offers GPU computing with declarative grammar for application programmers to build high-performance visualization systems. P4 leverages runtime code generation to create WebGL programs that effectively utilize the GPU to accelerate both data transformation and visualization rendering, providing performance that is an order of magnitude faster than the current state of the art. In this paper, we present P5, a visualization toolkit that combines declarative visualization and GPU computing for progressive data analysis and visualization. The design of P5 is motivated by three major goals.

Progressive Parallel Processing. Parallel computing and progressive processing are two promising approaches for supporting big data analysis and visualization, and they can also be combined to complement each other. By exploiting parallel computing for progressive processing, visualization systems can deliver faster results at each progression, accelerating the overall analysis and exploration process. Additionally, enabling progressive processing for GPU computing allows handling data that is larger than the GPU memory capacity. A framework is needed to effectively combine progressive processing with GPU computing.

Declarative Progressive Visualization. Implementing either parallel processing or progressive analytics alone for data visualization is already a nontrivial task. Coupling these two methods for building a visualization system requires significant programming and software engineering effort. Hence, a visualization toolkit with an easy-to-use application programming interface (API) can be helpful for implementing progressive and parallel data transformations and visualizations in analytics systems. This relieves the huge technical burden on the designers and programmers for implementing both parallel processing and progressive visualization from scratch.

Interactive Progressive Analysis. Enabling interactive visualization for progressive analytics is important for allowing users to analyze the intermediate results. For using statistical computations to progressively analyze big data, individual data records are discarded after each progression; as a result, direct data filtering cannot be used for supporting interactive visualizations. To address this issue, visualization toolkits should provide functionality for indexing and saving the data structures required by the user-specified interactions.

To meet these goals, we first contribute a visualization framework that seamlessly integrates GPU computing with progressive processing. As a web-based visualization toolkit, P5 leverages and extends P4's GPU-based data transformation and visualization capabilities. This allows P5 to effectively facilitate progressive parallel processing. Second,

- *Jianping Kelvin Li is with University of California at Davis. E-mail: kelli@ucdavis.edu.*
- *Kwan-Liu Ma is with University of California at Davis. E-mail: ma@cs.ucdavis.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

we contribute an intuitive API with declarative grammar for specifying progressive data transformations and visualization operations, both of which effectively run on the GPU. Moreover, P5 automatically generates indexed data structures for supporting interactive visualization, so users can interact with the visualization of the intermediate results in the progressive analysis workflow. We demonstrate the usefulness and advantages of P5 through a variety of example applications. In addition, we run performance benchmarks to show that P5 can drastically improve the speed of progressive visualization and provide interactive performance for analyzing intermediate results.

2 Related Work

P5 is related to research work in progressive analytics, visualization toolkits, and interactive visualization systems.

2.1 Progressive Analytics

In the field of computer science, progressive methods are commonly used for trading off computation latency against quality of results, where the results can be incrementally refined or improved over time. Progressive refinement methods [12, 43] were introduced in the area of computer graphics for supporting interactive rendering, where similar approaches have been adapted for scientific visualization [18, 22]. In the area of databases, a long-running thread of research in approximated query processing [4, 21] has provided a basis for progressive data analysis. With databases supporting approximated query processing, progressive data analysis systems can trade off accuracy for response time by returning partial results that are progressively improving. Running confidence intervals are usually computed and provided with the results. Fisher et al. [17] showed that incremental visualization of progressive aggregation results with confidence intervals can allow users to effectively make early decisions when exploring large datasets. Schulz et al. [37] presented an enhanced incremental visualization model for generalizing the progressive process of using partitioned data and visualization operators to facilitate intermediate visualization updates. Zgraggen et al. [46] studied the effect of progressive visualization compared to a blocking condition with significant delays and an ideal condition with almost no delay. They showed that users performed equally well with progressive visualizations and ideal condition with almost no delay in discovering insights, but the blocking condition with significant delay adversely impacts the analysis process.

By extending the concept of progressive data analysis for visual analytics [13], progressive visual analytics (PVA) provides visualization of the intermediate results and allows analysts to interact with the intermediate results for steering the analysis process. Steering typically includes changing the parameters of the analysis methods and visual encoding of the visualization, as well as stopping or restarting the analysis. Recently, researchers have developed several PVA systems for different analysis tasks and studies. Stolper et al. [40] adapted the SPAM [5] algorithm in a PVA system for exploring the correlations in electronic medical records. Badam et al. [6] presented InsightFeed, a PVA system for exploring social network feeds, and conducted user studies to investigate the tradeoffs in user interface design for supporting progressive visual analytics. Pezzotti et al. [31] developed DeepEyes, a PVA system that provides visualizations for supporting the design of neural networks during the training process. Most of these systems followed the design guidelines provided by Stolper et al [40], which are useful for designing the analytics and visualization components in PVA systems. Based on these guidelines, the analytics component should provide incrementally refining partial results and allow users to focus on subspaces of interest as well as to ignore irrelevant subspaces. On the other hand, the visualization component should indicate new results, support on-demand updates, minimize distractions due to updating results, and provide interfaces to specify problem subspace. These design guidelines also provided a basis for designing P5's API with the goals of supporting progressive data processing and visualization.

2.2 Visualization Libraries and Toolkits

Many visualization libraries and toolkits have been developed for designing visualization applications and building visual analytics systems. High-level libraries, such as ggplot2 [44], D3 [7], and Vega/Vega-Lite [35, 36] provide declarative grammars to reduce the technical burdens for crafting visualizations. Declarative grammars can decouple visualization design from execution details and defer control flow concerns to the runtime, which allow programmers to focus on application-specific design decisions [20]. However, these libraries focus on providing an easy-to-use programming interface, but do not offer support for processing and visualizing large data. Application developers need to incorporate high-performance computing techniques for handling large datasets, which undo the benefits of declarative visualization specification. On the other hand, low-level graphics libraries, such as OpenGL and VTK, provide fine-grained control and performance, but are tedious for building advanced visualization systems and data analytics applications. Besides system performance, these libraries do not provide support for progressive data processing and visualization. This makes building visualization systems that incorporate both high-performance computing and progressive analytics workflows challenging.

Only a few researchers have contributed to research and development of high-performance toolkits for information visualization. McDonnel et al. [29] presented a visual programming environment with graphical user interfaces for composing GPU shaders to create data visualizations. However, the flexibility for creating customized visualizations is limited. Ren et al. [34] developed Stardust for providing a programming interface similar to D3 which uses WebGL to render large data. While Stardust only focuses on improving rendering performance, it does not provide support for accelerating data transformations. Fekete et al. [16] presented ProgressiVis, a Python toolkit for providing a progressive computation paradigm for implementing exploratory data analysis applications. However, ProgressiVis only uses sequential computations and provides no direct support for interactive visualizations. To provide high performance for progressive data analysis and visualization, we combine GPU computing and progressive processing in P5. In addition, we design an easy-to-use API with declarative grammar for specifying progressive analysis workflows. P5 provides common data transformations and visualizations for streamlining the implementation of progressive visualization systems.

2.3 Interactive Visualization Systems

Supporting interactive analysis is important for visualization systems. Many visualization systems use a server-client architecture for processing queries using server-side databases and send the results to the client for visualization. For example, commercial products such as Spotfire [2] and Tableau [1] use database systems in cloud computing infrastructures to process queries for supporting interactive visualizations. In this approach, every user interaction requires a round trip to get the results from the database, where the associated queries need to be evaluated on the entire dataset. To reduce system latency for interactive visualization, data indexing techniques, such as data cubes [19], can be used to compute data aggregates based on the visualized data dimensions. The interactive scalability is not limited by data size but by the chosen resolution of the visualizations. Multi-scale data cubes [41] can be used to support interactive visualizations of multiple resolutions. Specialized data cubes can be designed for specific data types. Nanocubes [26] use hierarchical structures and compact indexes for interactive visualizations of spatiotemporal data. Methods based on data cubes typically create precomputed data aggregates stored in system memory to support real-time interactive visualizations. However, the storage size of data cubes increases exponentially with the number of data dimensions and polynomially with the resolutions of the dimensions. When the size of the data cubes is large, processing the data cubes can incur high system latency. In addition, the required storage size for the data cubes can be larger than the capacity of the system memory. To address these issues, Liu et al.'s imMens [28] create precomputed aggregates as dense data cubes based on the linked data dimensions in the visualizations, which decompose a full data cube into several smaller data tiles to reduce storage size. The precomputed data tiles are then stored in GPU memory to leverage GPU computing for parallel processing, allowing imMens to support interactive visualizations of larger datasets with low latency. Beside storage size

and processing speed, methods based on data cubes require preprocessing, which can take a long time before users can start exploring the data. In our approach, P5 leverages GPU computing to automatically generate small data cubes based on the interactions defined by the declarative grammar. P5 also automatically updates the data cubes during progressive data processing. This enables cold-start exploration of large datasets without precomputation and leverages GPU computing to accelerate data processing for supporting interactive visualizations.

3 Design

In this section, we first present P5's progressive visualization model and system framework that enable progressive processing with GPU-based parallel data transformations and visualizations. Second, we introduce P5's API and declarative grammar for designing and specifying progressive processing pipelines. We also describe our methods for facilitating interactive visualization during progressive data processing. Last, we provide the details for our implementation of P5.

3.1 Progressive Visualization Model

P5's progressive visualization model extends the information visualization Reference Model [8]. A conventional visualization pipeline based on the Reference Model is shown in Fig. 1(a), where analytical processing includes data transformations and statistical analysis methods, and visualization rendering includes visual mapping and view transformation. With such conventional pipeline, users need to wait for the processing and rendering of the whole dataset to be completed before they can see and analyze the results. For large datasets, the wait time can be very long which may hinder effective visual analysis process [9]. To provide a progressive analysis workflow, P5 extends the visualization pipeline based on the Reference Model to adopt the progressive analytics paradigm as shown in Fig. 1(b).

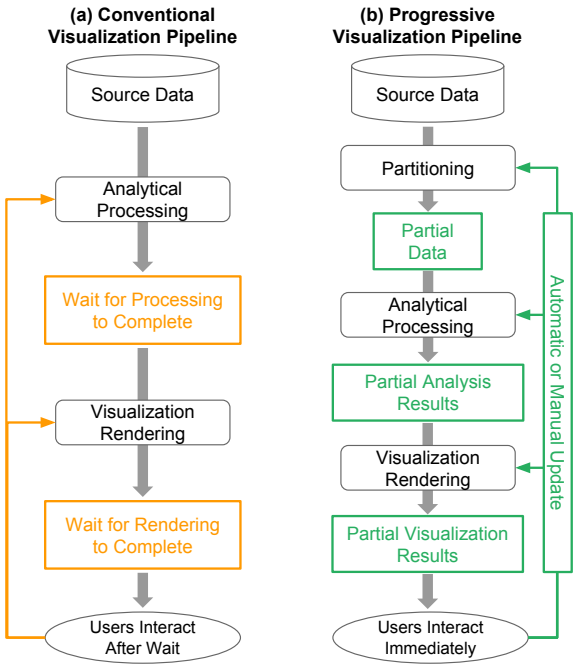

Fig. 1. Conventional pipeline (a) versus progressive pipeline (b) for data visualization. The convention pipeline requires users to wait for analysis and visualization operations to be completed on the whole dataset before interaction, while the progressive pipeline processes data in chunks and provides incrementally refining visualization of results at interactive speed.

In our model of progressive pipeline, a partitioning operation is first performed to obtain data chunks from the data source (e.g., files in local hard disks or server). Analytical processing is then performed on the data partitions to produce partial analysis results, which can be visualized and incrementally refined. To support different types of progressive analytics applications, P5 allows the progressive pipeline to operate with three different modes: automatic, manual, semi-automatic. In automatic mode, the pipeline automatically processes the next data partition after the visualization of the current result is completed. In semi-automatic mode, analytical processing is progressing automatically, but visualization rendering is performed manually. An explicit signal is needed to be received by the pipeline to visualize the accumulated partial results. In manual mode, a explicit signal is needed to perform both analytical processing and visualization rendering. The operations for partitioning, analytical processing, and visualization are parallelized via GPU computing and can be adjusted or changed during progressive processing.

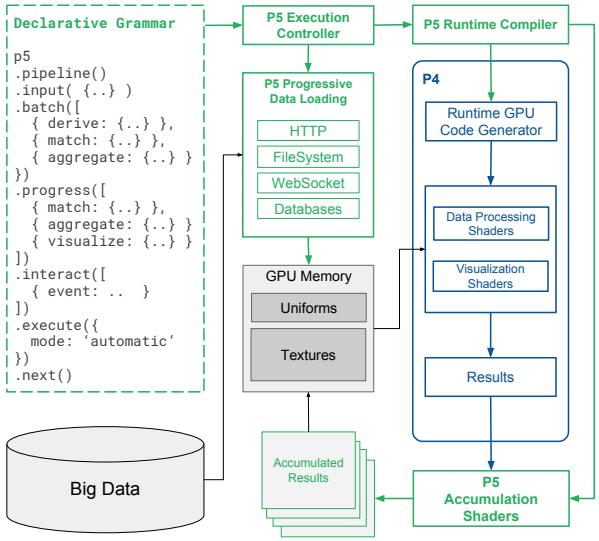

Fig. 2. System architecture and components of P5 for enabling progressive parallel processing.

3.2 System Framework

Based on our progressive visualization model, P5 extends P4's framework to facilitate progressive parallel processing. Fig. 2 shows the system framework of P5. P5 extends the declarative grammar of P4 for specifying data transformations and visualizations, and it uses P4's GPU code generator to create and update GPU shader programs at runtime for progressive processing. The execution controller in P5 executes all the operations listed in the declarative specification based on the *pipeline mode* (automatic, manual, or semi-automatic). For data partitioning, P5 provides a set of data loading methods (HTTP, file system, Web Socket, and database connectors) for progressively loading data from external sources. These methods obtain data partitions from the data sources and store them in the GPU memory. At runtime, P5 creates GPU programs to perform all the specified data transformations and visualizations on the GPU. P5 then uses the GPU to merge the new results with the accumulated values. After accumulation is completed, P5's execution controller loads the next data partition for progressive parallel processing.

3.3 Application Programming Interface

P5 provides an intuitive API for specifying and controlling the progressive workflow. The syntax and high-level operations of P5's API are shown in the left of Fig. 2. Following the convention used in P4's syntax, progressive operations are specified as a *pipeline* in P5. The input data need to be first specified for obtaining data partitions by progressively loading from a data source. The *batch* operation is used for specifying statistical computations and data transformations to process each data partition, and the results of the *batch* operation are automatically accumulated to generate the intermediate results. The *progress*

operation is used for specifying data transformations and visualizations to analyze the intermediate results during progressive processing. User interactions to the visualizations can also be specified for selecting, linking, and correlating results in different visualization views. The specified *pipeline* can be executed in automatic, semi-automatic, or manual mode. At each iteration, the *progress* operations can be changed to visualize and analyze different aspects in the intermediate results. However, the *pipeline* needs to be restarted if any changes were made to the *batch* operations. This is because P5 does not keep the previous data items in memory, and thus it needs to redo all the computations from the beginning for making sure the accumulated results are correct.

Fig. 3 shows an example of a P5 *pipeline* for progressive analysis of a dataset that contains information about newborn babies and their parents. This *pipeline* progressively loads data partitions from a local file. The *batch size* is for controlling the size of data partitions in each progressive loading. When the input data is from a database or server, a number of data rows or items can be specified for progressive loading instead of the batch size. In the *batch* operation, data filtering and aggregation are performed on the data partitions to obtain the counts of newborn babies for each combination of the ages of the parents. The results are then accumulated for visualizing as a 2D heatmap based on the *progress* operation.

```
p5.pipeline()
.input({
  source: 'file://.../data.csv',
  batchSize: 500000,
  type: 'text/csv',
  delimiter: ','
})
.batch([
  {
    match: {
      MotherAge: [18, 50],
      FatherAge: [18, 70]
    },
    aggregate: {
      $group: ['FatherAge', 'MotherAge'],
      $collect: {
        Babies: {$count: '*'}
      }
    }
  }
])
.progress([
  {
    visualize: {
      mark: 'rect',
      x: 'MotherAge',
      y: 'FatherAge',
      color: 'Babies'
    }
  }
])
.execute({mode: 'automatic'})
```

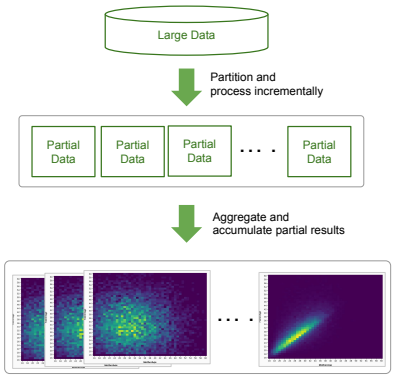

Fig. 3. Declarative specifications in P5 for progressively processing the baby birth dataset, in which data filtering and aggregation are performed to visualize the results in a 2D heatmap.

3.4 Signals and Controls

To control the progressive processing workflow, P5's API provides signals and control operations for managing the states and execution of the *pipelines*. A P5 *pipeline* can be started to process and visualize data by calling the *execute* function, for which the execution mode (automatic, semi-automatic, or manual) can be specified. A new batch size or time bounds for progressive processing and visualization can be provided for controlling the frequency of the updates. With automatic mode, the *execute* function starts to progressively perform the specified operations until all data are processed. *Pipelines* with automatic mode can be controlled by the *pause* and *resume* functions. A *next* function is provided for updating the visualization views for semi-automatic and updating both processing results and visualization views for manual mode. In addition, event signals, *onEach* and *onComplete*, are triggered at end of the each progression cycle and when all the data partitions are processed, respectively. Event handler functions can be defined to perform custom operations or other application specific tasks based on these event signals. For the *onEach* event, the number of processed data records and the status of progressive processing are passed to the event handler.

3.5 Data Transformation

Common data transformations supported in P4, such as deriving new values, filtering, binning, and aggregating, can be specified in P5's *pipeline* for both the *batch* and *progress* operations. Since the progressive process needs to be restarted if any changes were made to the *batch* operations, we have extended P4's declarative grammar for aggregation to provide more flexibility and better support for progressive data analysis. Fig. 4 shows P5's extended grammar for specifying binning and aggregation operations.

P4 Specification

```
{
 $aggregate: {
   $group: ['MotherAge', 'MotherRace'],
   $collect: {
     sumWeight: {$sum: 'BabyWeight'},
     avgMotherAge: {$avg: 'MotherAge'},
     count: {$count: '*'},
     ...
```

P5 Specification

```
{
  $aggregate: {
    $group: ['MotherAge', 'MotherRace'],
    $include: ['*'],
    $exclude: ['Father*'],
    $calculate: ['avg', 'sum', 'count']
  }
}
```

Fig. 4. P5's extended syntax of data aggregation for allowing multiple attributes and measures to be included in the results.

Using P4's declarative grammar, users need to explicitly specify which data attributes and what metrics (e.g., count, sum, or mean) to be collected for the aggregation operation. With P5's extended syntax, the *include* and *exclude* clauses can be used to select or skip any data attributes, and the *calculate* clause can be used to calculate all the metrics using the selected attributes. This can avoid restarts of the progressive process caused by changes to the aggregation specification for the *batch* operation, because users can select all the attributes and metrics of interest. The process of progressive data analysis only needs to be restarted when users significantly changed their ways to analyze the data by using a different set of attributes for filtering and aggregation. The latency overhead to the parallel aggregation due to supporting this extension is small, because calculating additional metrics via GPU computing is trivial.

3.6 Visualization

P5's API adopts and extends P4's declarative visualization grammar for supporting progressive data visualization. Data processing results can be easily mapped to visual encoding channels, such as color, opacity, width, height, and position, for generating various visualizations. As not all analytic methods can be used for progressive data analysis [40], not all visualization types are suitable for progressive or incremental visualization. Therefore, P5 only uses a subset of the supported visualizations in P4. Fig. 5 lists all the visualizations supported in P5 that can be used for plotting numeric, categorical, temporal, and geographical data. In addition, P5 extends P4's visualization grammar for specifying multiple visualizations as faceted views.

Fig. 6 shows an example of faceted views as rows for visualizing the aggregation results of a multivariate time-series dataset. In the facet specifications, *variables* can be defined for generating rows or columns of visualizations based on the *layout* parameter. The faceted views can be sorted based on calculating different measures for the visualized items, such as average values and variances. In this case, three values are defined for the *metrics* variable for plotting three area charts ordered by their variances. During progressive processing, the sort orders are updated at each iteration to inform the most important view.

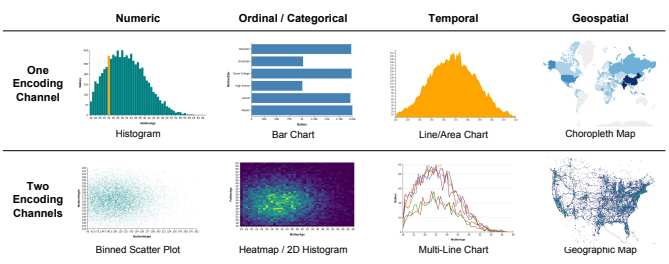

Fig. 5. P5's visualizations for supporting progressive data analysis and visualization with different types of data.

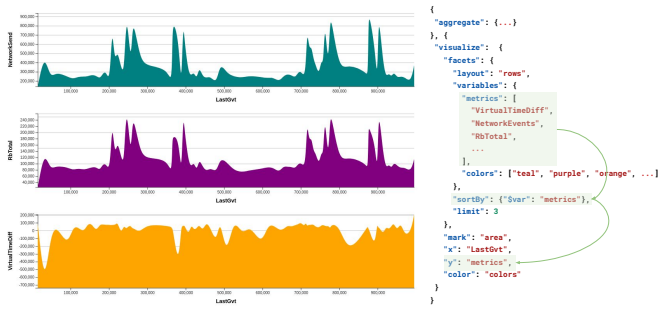

Fig. 6. Declarative specification for visualizing multivariate time-series data in faceted views showing the top 3 user-defined variables ordered by their variances.

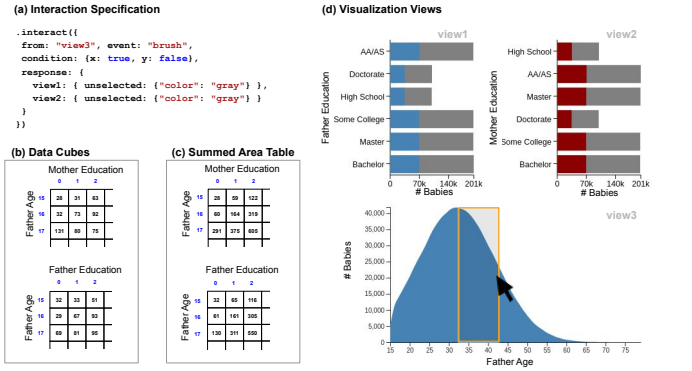

Fig. 7. Interaction specification (a) for connecting the visualization views (b) based on the selection of the user. P5 automatically generates the data cubes (c) for supporting interactive visualization during progressive processing. Summed-Area Table (d) can also be computed from the data cubes to support interactive performance for higher resolutions.

3.7 Interaction

Interactive visualizations and multiple coordinated views are useful for visual analysis of multidimensional data. Users can gain insights by making a selection in one view to filter data and show correlations in the other views. However, data filtering cannot be used directly with progressive processing because individual data items are discarded after the statistical computations are completed. To support interactive visualization for progressive processing, P5 automatically generates data cubes based on the visualized data dimensions for effective data filtering. P5's API supports specification of user interactions for linking visualizations and highlighting selections. Fig. 7 shows an example for specifying a brushing-and-linking interaction for selecting a range in the area chart to highlight the corresponding data in the two bar charts. Because the visualizations are linked at the data level, selection in one view is associated with the visualized data in all the views. In the interaction specification, users can just specify the changes in visual channels for updating the targeted views (view1 and view2) based on the event (brush) from the source view (view3). During progressive processing, P5 automatically generates the required data cubes based on the user specifications for interactive visualization. The generation of the data cubes is accelerated by GPU computing. To reduce the memory footprint, P5 creates small data cubes for only the data dimensions associated with the specified interaction, instead of creating a large data cube with all the visualized dimensions. For the example shown in Fig. 7, two 2D data cubes, instead of a 3D data cube, are created for the interaction that updates the two bar charts based on the selection on the are chart. This reduces the size of the data cube from x^3 to $2x^2$, with x representing the resolutions of the data dimensions. As the number of data dimensions associated with the interaction increases, this technique can help saving more memory space.

By default, P5's execution controller pauses the *batch* operations during user interactions. Because both the *batch* operations and the processing of data cubes run on the GPU, concurrent execution results in high latency for interactive visualization. To run *batch* operations and support interactions at the same time, P5 can be configured to extract the data cubes from GPU memory to system memory and uses the CPU for processing the data cubes. However, this increases the time for *batch* operations in each progression due to the overhead of data transfers. To allow faster processing of the data cubes on the CPU, we convert the data cubes from a dense format to a sparse format when extracting them from the GPU memory. This reduces both the memory footprint and the time needed for processing. To support interaction with higher resolutions, summed-area tables [14] can also be computed from the data cubes to ensure interactive speeds. In a summed-area table, each value at each location is the inclusive sum of all the values in the data cube up to that location, as shown in Fig. 7 (c) and (d). The summed-area table can then be used to quickly return the values for rectangular selections by subtracting the beginning indices from ending indices, reducing the computation complexity from $O(x^2)$ to $O(x)$ in this example. This method can be used for supporting interactions with more visualized data dimensions as well.

3.8 Outputs and Exports

To allow P5 to be used with other visualization libraries and toolkits, our API supports exporting the immediate results of progressive processing. P5 can output the results in JSON and TypedArray formats, as it supports these two formats for data inputs. In addition, P5's API allows callback functions to be defined with the interaction specification for exporting the results on demand. As shown in Fig. 8, an interaction can be specified with a callback function that exports the results of the progressive aggregation. For interactions that are not linked to other views, we can define the *brush* within the visualization, and we can just define one interaction for the faceted views since the selections on their x-axes are the same. In the callback function, the result exported from the *pipeline* is provided as the input. In this example, P5 exports the progressive aggregation result according to the selection on the timeline charts. The result is then used as the input for D3 to generate the customized circular visualization on the right panel. This allows developers and programs to use P5 with other visualization libraries and web technologies for building custom visualization systems. In addition, multiple outputs can be computed and tagged in the *batch* operation for creating different progressive visualization views. Fig. 9 shows an example for generating multiple outputs in the *batch* operation for creating different views with progressive visualization.

3.9 Implementation

As a web-based toolkit that leverages GPU computing, the implementation of P5 is based on JavaScript and WebGL 1.0. P5 leverages P4's capabilities for performing data transformation and visualization operations on the GPU, as well as for managing data in the GPU memory, including progressively loading data chunks and storing the accumulated results. By storing data in WebGL floating-point textures, P5 can support 24-bit precision with values ranging from 2^{-127} to 2^{127}. Categorical attributes are managed as integers for processing efficiently with GPU computing. For progressive processing and visualization, we implement a new WebGL shader program for P5 to efficiently accumulate the data processing results in each progression. For partitioning and loading data, P5 provides various methods for progressively retrieving data via various protocols, including FileAPI, HTTP, WebSocket, and MySQL database. For the remote methods that require a web server, P5 provides server-side modules written in NodeJS for programmers to use in their server-side programs, which can be used with the progressive pipelines created by P5 in web applications. Alternatively, programmers can use P5's protocol for progressive data loading in other server programs that are not based on NodeJS, such as Python and C++.

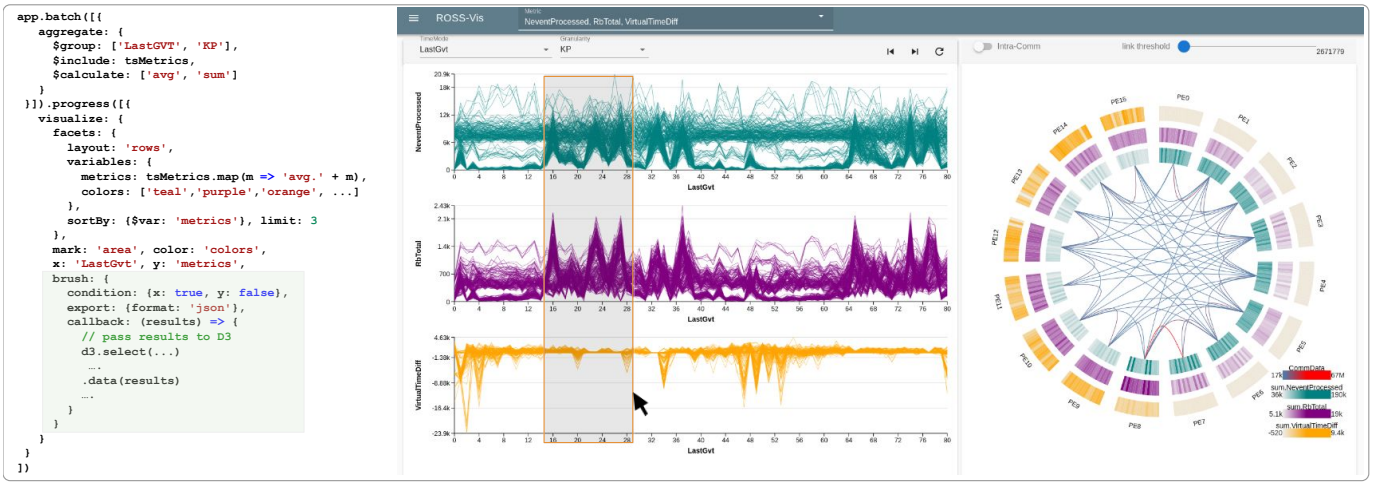

Fig. 8. A progressive analytic application implemented by P5 for analyzing large multivariate time-series data using faceted views and interactive visualizations. P5 allows custom callback functions to be defined in the user interaction for exporting the results of progressive processing to create customized visualizations, such as the circular chart on the right.

4 Example Applications

We have built several progressive visualization applications using P5 over the course of its development. Here we present three of these applications to demonstrate P5's applicability and usefulness.

4.1 Multivariate Time-Series Analysis

Parallel discrete-event simulation (PDES) is a cost-effective and useful tool for modeling and researching in many areas, ranging from studies of complex physical phenomena to the designs of supercomputers. Therefore, it's important to ensure the efficiency of PDES to minimize time and energy. Optimizing the performance of PDES typically requires post-hoc analysis on large multivariate time-series data collected from simulations. Due to the large size of data generated by PDES, interactive analysis of the whole dataset is difficult. By collaborating with PDES researchers, we have designed and built a progressive visual analytics system using P5 for analyzing the PDES data collected from the the ROSS simulator [10]. ROSS uses Message Passing Interface (MPI) for distributed computing, in which each processor entity (PE) is assigned to a MPI job. Each PE contains a fixed number of kernel processes (KP) with each KP managing a group of logical processors for simulating different items in a simulation model. For each entity, 16 performance metrics are collected over the simulation time.

The user interface of our system and the visualizations with their declarative specification are shown in Fig. 8. The line charts on the left panel shows the top 3 of the 16 metrics ordered by the overall variances over time. As ROSS uses the last global virtual time (LastGVT) to keep track of the time in the simulations, we use LastGVT as the x-axis for all the line charts. By modifying the *$group* parameter in the aggregation specification, the granularity of the line chart can be changed to show each line as either a PE, KP, or LP based on the selected level of detail. In addition to showing the temporal behaviors of the top three metrics, our system shows the communication patterns between PEs using hierarchical circular visualization methods [25] in the right panel of our user interface. While all the data processing and visualization of the line charts are implemented using P5, the hierarchical circular visualization is implemented using D3. Our system leverages P5 to progressively process the large time series data and export the results based on user selections. The specification of the user interaction is highlighted in green in Fig. 8. According to the operations in the callback function defined in the interaction specifications, once the user makes a selection of the line charts, the results corresponding to the selected time range are exported from P5 to D3 to create or update the circular visualization in the right panel. In the hierarchical circular visualization, the top three performance metrics with the highest variances over time are stacked on the circular visualizations from the inner to outer layers, which is arranged by the order of the associated PE and KP to show the distributions and correlations of the communication patterns.

In addition to progressively analyzing PDES data, our system can also receive data streaming from running simulations via WebSocket. This allows PDES developers and users to monitor simulations and detect performance problems, so they can stop problematic simulations and restart the simulation with better configurations to avoid performance bottlenecks. This example application shows the usefulness of P5's declarative grammar for specifying aggregation and creating visualizations in faceted views. It also shows that the user-defined interactions with callback functions are helpful for building advanced visualization systems for progressive analysis of large time-series data.

4.2 Progressive and Interactive Visual Analysis

Coordinated and linked views of histograms and summaries are useful for exploring multidimensional data. To demonstrate P5's capability for building this type of application, we use P5 to reproduce the interactive visualizations used in the imMens system by Liu et al. [28]. The imMens system provides linking-and-brushing interactions for visual querying of big data in binned scatter plots and histograms. To generate scalable visualizations and to support user interactions, imMens uses database systems to generate precomputed data tiles based on the visualized data dimensions in all of the views. However, the disadvantage of this approach is that the visualizations cannot be changed to analyze other data dimensions, because it relies on data preprocessing.

With P5, the capabilities of progressive and GPU-based data processing can be leveraged to incrementally perform parallel aggregation and visualize different data dimensions without relying on database systems or data preprocessing. To compare with imMens, we use the same dataset (Brightkite user check-in [11]) used in imMens, and we generate the same set of views with the same visual resolutions, which are a binned geographic map and three bar charts. Fig. 9 shows the declarative specifications and the visualization views. The input data source, data schema, and view dimensions are specified in (a). The *batch* operations are specified in (b) that progressively derives new attributes (months, days, and hours) and performs aggregation and binning based on these new attributes. The results of progressive processing are visualized according to the specifications in (c), which generate the views in (d). In addition, the interaction is specified in (e): highlighting the corresponding data selected by brushing the map in the three bar charts. All the data transformation and visualization operations are accelerated by the GPU and are progressively updated.

As this example application shows, P5 can be used to implement an advanced visualization system to interactively explore big data with approximately 100 lines of code of declarative grammar. Compared

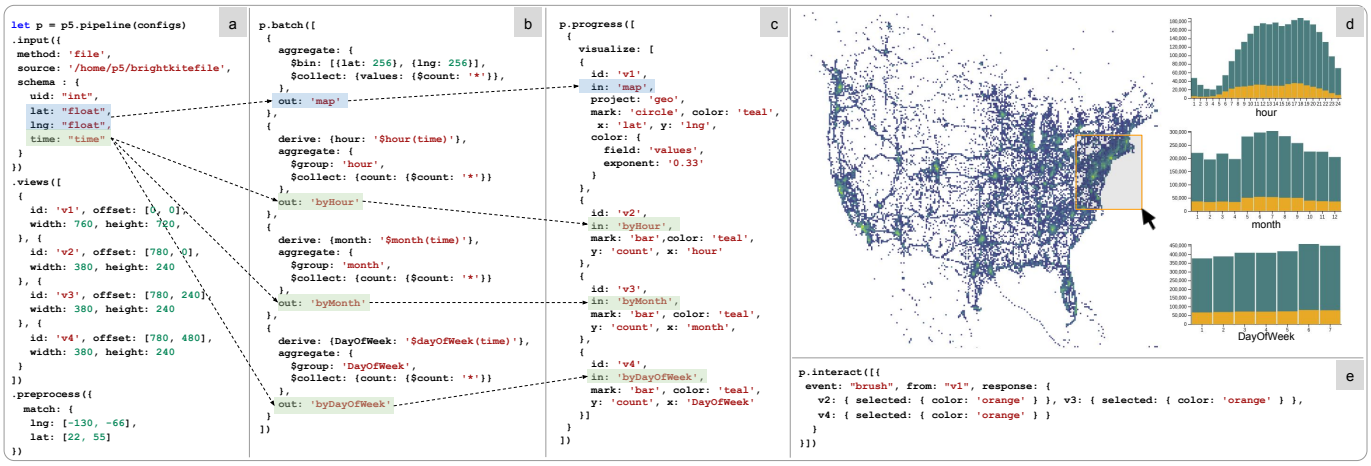

Fig. 9. We use P5 to reproduce the dashboard used by imMens for visual exploration of the Brightkite dataset. The specifications of the data and view (a), *batch* operations (b), visualization of the results (c) in P5's declarative grammar are used to generate a set of coordinated and linked views (d) with the brushing-and-linking interaction specified in (e).

to imMens, which is hard-coded in WebGL, P5 provides flexibility to explore big data through declarative specifications and automatic generation of GPU programs and data structures. With the progressive analysis provided, analysts can easily adjust or change the specifications of the data transformations and visualizations after inspecting the initial results. For example, the analysts can change the visualization specification to show the number of check-ins based on the day of the week for exploring the weekly check-in patterns of the Brightkite users.

4.3 Cold-Start Data Exploration

In parallel to the development of P5, we have created P5 Play, a web application to progressively analyze and explore big data using P5's declarative grammar. The user interface of P5 Play is shown in Fig. 10, which is composed of a data panel (a), a control panel (b), an editor (c), and a visualization dashboard (d). P5 Play leverages the HTML5 File API [3] for loading data from files in local hard disks, which does not require data uploads via the Internet. Currently, P5 Play can support JSON and CSV file formats. The data panel allows users to select a data file and lists all the data attributes in the selected file. The editor is used to provide the declarative specifications to progressively process the data and display the visualizations of the incrementally refining results in the dashboard. The control panel manages the progressive workflow, where users can reset, pause, or proceed to the next progression. One of the three modes for configuring P5 (automatic, manual, or semi-automatic) can be selected. The two progress bars show the percentage of the data processed (blue) and visualized (green). This control panel is implemented using P5's signal and control functions as described in Sect. 3.4. Once the data file is loaded and the declarative specifications are inputted, users can use the control panel to start exploring the data. P5 Play allows a cold-start exploration of big data without the need of data preprocessing or reformatting the data into a database system. By providing progressive data analysis, users do not need to wait until data processing and rendering are completed for the entire dataset. Instead, users can analyze the intermediate results, adjust or change the visualization specifications, and restart the progressive analysis process with new analytical methods, if needed. With exploratory visual analysis, this allow users to create and verify hypotheses at a much faster rate.

5 Performance Benchmarks

Here we provide the performance benchmarks to evaluate P5. All our benchmarks were performed in Google Chrome 73.0.3683.75 (64-bit) on a desktop computer with a quad-core 3.6 GHz Intel i7-4790 CPU and a Nvidia GTX Titan graphic card connected by PCI Express.

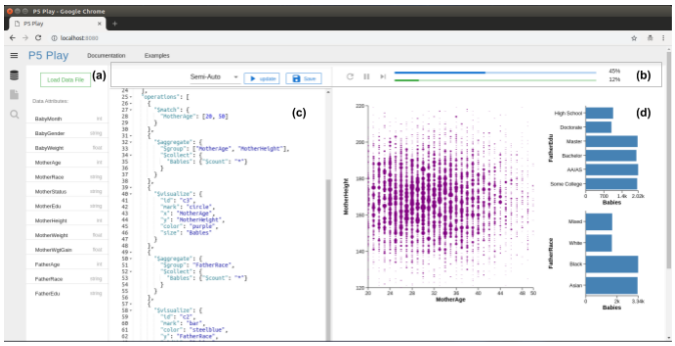

Fig. 10. P5 Play allows users to progressively analyze multidimensional data using declarative visualization grammars. P5 Play's user interface consists of a side panel (a) for selecting data files and listing the data attributes, a control panel (b) for starting and pausing progressive operations, a text editor (c) for editing declarative specifications, and a dashboard (d) for displaying the output visualizations.

5.1 Progressive Analysis Performance

To measure P5's performance for progressive data processing and visualization, we generate a dataset with 100 millions of data records with random samples drawn from normal distributions. Each data record contains 3 integer attributes, 4 float attributes, and 4 categorical attributes. For loading the data, we use a server-client setting, where the generated dataset is stored in the server and the client progressively loads the data chunks from the server via HTTP request. In this benchmark, we use the data transformation and visualization operations in Fig. 3. To show P5's advantage over existing visualization libraries for progressive visualization, we compare P5 to D3 [7] and Stardust [34]. While D3 provides a flexible API with declarative grammar for rendering visualizations on SVG, Stardust leverages WebGL for rendering with a similar API to D3. To implement progressive visualization using D3 and Stardust for our benchmark tests, JavaScript's Array functions are used to implement the data transformation operations, and the APIs of D3 and Stardust are used to visualize and upate the results. Since P5 uses TypedArray as the default data type for the input while D3 and Stardust cannot work with TypedArray, the benchmark tests for D3 and Stardust use JSON as the data type of the input, and both JSON and TypedArray are used for P5. Fig. 11 shows the benchmark results of D3, Stardust, and P5 for progressive processing and visualization of

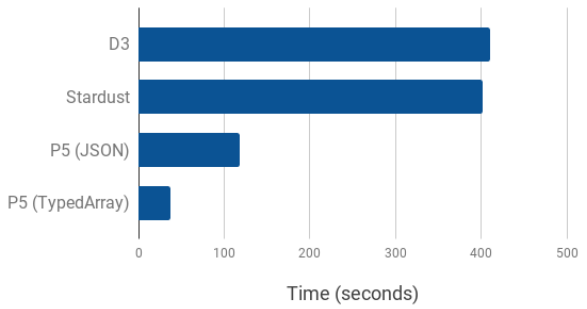

Fig. 11. Completion time for D3, Stardust, and P5 to progressively process and visualize a dataset with 100 million data records.

100 million data records.

As shown in the results, D3 and Stardust have similar completion times. Since Stardust only utilizes the GPU for rendering but not data processing, it performs data transformations on the CPU and transfers the results to the GPU for visualization rendering. Although Stardust is faster than D3 for rendering, the overhead due to the data transfers between the GPU and CPU makes Stardust have a similar completion time comparing to D3. With P5, data transfers between the GPU and CPU are also required, but both data transformations and visualization renderings are performed on the GPU, which drastically improve the overall performance. Comparing to D3 and Stardust, P5 with JSON is about 3.5 X faster, and P5 with TypedArray is 10 X faster. Using TypedArray as the data type can make data transfers between GPU and CPU more efficient, allowing P5 to achieve better performance comparing to JSON. Besides performance, both D3 and Stardust require significant programming efforts to implement the operations for data partitioning, progressive data loading, and visualization updates. In contrast, P5's API allows application programmers to easily specify these operations, providing both high productivity and performance.

In addition to the overall completion time, we run a performance test to measure the time needed by each progression with different batch sizes (100K, 500K, 1M, and 2M). Each test runs for 10 progressions for each batch size and measures the average time needed to complete all the operations. The results are shown in Fig. 12, where we can see that P5 can provide a better response time for progressive processing. The results also indicate that P5 can not only provide a better performance for progressive data analysis but also can better support streaming data visualizations with a larger data and at a faster rate. As we can see from this benchmark test, P5 can support a streaming rate of one million data items per second, with each data item containing 10 data dimensions.

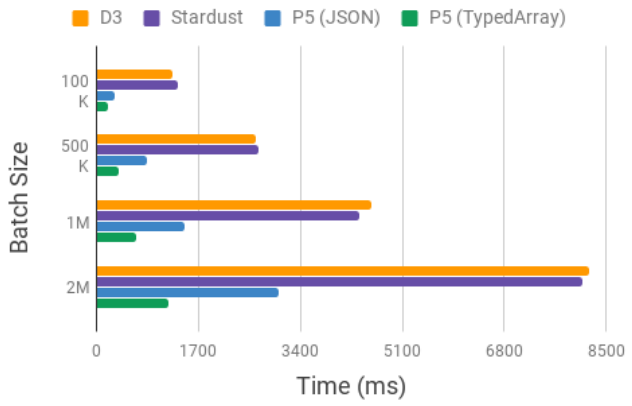

Fig. 12. Average time (ms) for D3, Stardust, and P5 to progressively process and visualize data with different batch sizes.

5.2 Interactive Performance

For evaluating the interactive performance, we compare the interaction latency of P5 with imMens [28]. As shown in Sect. 4.2, to explore the Brightkite dataset, P5 can be used to implement the same visualization views with user interactions as imMens. Here we compare the brushing-and-linking interaction for P5 and imMens. With the brushing-and-linking interaction, the user makes a rectangular selection on the geographic map, and then the corresponding statistical values are highlighted in the three bar charts. We add instrumentation in P5 to measure the interactive performance for each user interaction, and we use the Frame Rate Meter in Google Chrome to obtain the average frame rate for imMens. For both P5 and imMens, we perform 20 brush selections with 20 random rectangular areas and obtain the average frame rate. For P5, we measure the performance for the techniques using dense and sparse data cubes as described in Sect. 3.7. The benchmark results are shown in Fig. 13.

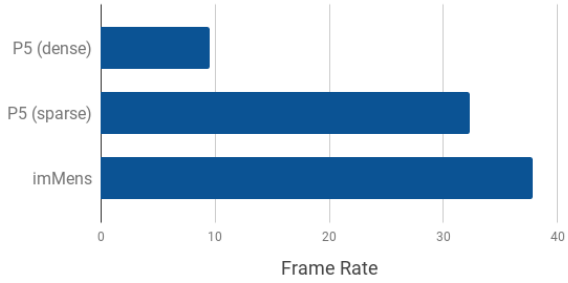

Fig. 13. Average frame rate for interactive visual querying of the Brightkite dataset with P5 and imMens.

As the results show, imMens achieved 38 frames per second while P5 achieved 9 and 36 frames per second with dense and sparse data cubes, respectively. It is expected that imMens has a better performance since it uses precomputed data cubes with the processing function hard coded in WebGL. For P5 with dense data cubes, because we use different WebGL textures for storing the progressive processing results and the data cubes for supporting the interactions, progressive processing needs to pause. This causes overheads to the system response time. When using sparse indexing, P5 computes the sparse data cubes from the dense data cubes and uses CPU for processing. This avoids pausing the progressive processing due to user interactions. As shown in the benchmark results, using sparse data cubes allows P5 to achieve better performance and is close to imMens' performance. As noted, the dataset used for this benchmark test has many zero values on the binned geographic map. This is the primary reason why P5 with sparse data cubes has a better interactive performance. For data that the sparse data cubes cannot take advantage of, we expect the performance difference between the sparse and dense data cubes to be small. In our current implementation of P5, Summed-Area Table cannot be used to support more two data dimensions with 2 or more views. Therefore, we did not test the interactive performance of P5 with the method based on Summed-Area Table. We expect it to have better performance than sparse data cubes, and we plan to improve our implementation in the future. Nonetheless, P5 with dense data cubes can achieve 9 frames per second, with the interactive latency being around 110 ms. According to the studies on the effect of interaction latency on visual analysis [8, 27], this level of interactive latency is still considered to be effective for supporting interactive analysis with visualizations. Moreover, the latency is not affected by larger data sizes since it is only affected by higher resolution, specified for user interactions. In addition, P5 provides more flexibility with declarative grammar and progressive visualization. Users can change the specifications to visualize and interactively explore different data dimensions.

6 Discussion and Future Work

The current implementation of P5 is an initial effort to develop a declarative visualization toolkit that is easy to use and high performance

for building progressive visual analytics systems. There remain many features that can be added to better support progressive data analysis and visualization. Here we discuss several possible extensions for enhancing P5 as well as the limitations that currently exist.

The statistical analysis and visualization methods provided by P5 can already support a large set of common visual analysis tasks for data exploration applications. However, exploratory data analysis often requires clustering and dimensionality reduction methods, such as K-Means and PCA, for exploring high-dimensional datasets. Many of these machine learning methods have high computational complexity, which causes long completion time for large datasets. Enabling user involvement in such methods with intermediate feedback and control can better facilitate interactive data exploration [30]. Recently, researchers from different domains, including visualization, data management, and machine learning, have begun to collaborate on addressing the challenges in progressive visual analytics [15]. It is expected that intermediate feedback and user involvement will be supported in more data analytics and machine learning methods, which can be used with progressive visualization. As P5 only supports common data transformations and statistical analysis methods, we plan to make P5 to be easily interfaced with other data analytics and machine learning libraries. When the results of incremental analytics methods can be fitted into GPU memory, P5 can retrieve the intermediate results from other libraries and incrementally render the visualizations. For example, the optimized t-SNE method [32] in the TensorFlow.js [39] library can be used for exploring high-dimensional data in 2D embeddings, and the intermediate results can be transferred to P5 for progressive visualization, which allows users to monitor the progress and adjust the parameters for steering the exploration process. When the size of the dataset or analysis results is large than the GPU memory capacity, P5 can still support data analytics methods that incrementally append results to visualizations. The visualizations of the intermediate results from these methods can be stored as images, and image scaling can be used to adjust the extents and size of the visualization according to the updates in the domains (e.g., maximum and minimum values) of the results. This method allows common visualizations with overlapping marks, such as a scatterplot, to be used for progressive visualization. However, this method cannot be used with some visualization types. For example, parallel coordinates plots are difficult to use for progressive visualization, because the change of maximum and minimum values on each of the parallel axes makes it difficult to scale the image to properly represent both the new and previously processed data. Unless the extents and domains of all the data are known in advance, such visualization types are difficult to use with progressive data analysis. In addition, we plan to make P5 work more efficiently with web-based data analytics libraries that use WebGL for GPU computing, such as TensorFlow.js. These libraries typically store their data and analytic results in WebGL textures. We can allow plugins in P5 for specifying the procedures to retrieve data from WebGL textures to P5's shader programs for visualization rendering. This can avoid the performance bottleneck due to the data transfer between the CPU and GPU when passing the analysis results from a WebGL-based data analytics library to P5.

While progressive visualization allows users to make early decisions and steer the analysis process, showing the uncertainty in the intermediate results is important. P5 currently lacks support for assessing and visualizing uncertainty. P5 only provides several metrics, such as sample means and variances, to programmers, and they need to implement their own mechanisms for showing uncertainty information in their applications. In addition, the current version of P5 only supports sequential sampling and partitioning for exploratory data analysis, which users typically do not know the structures and characteristics of the dataset for choosing a sampling method. For structured data files and database systems, random sampling method can be used, and the programmers can implement other sampling methods based on the needs of their applications. In our future work, we plan to provide more support in assessing uncertainty, such as calculating confidence intervals, as well as developing methods to visualize the uncertainty for better facilitating progressive data analysis.

Besides assessing uncertainty, P5 also has limitations on specifying interactions on multiple visualizations. P5's API can be used to specify interactions that link one view to many views (e.g., the example applications in Sect. 4.2). However, interactions that link many views to many views (e.g., linked scatter plot matrices) cannot be defined via P5's API in one specification. Specifications of such interactions require multiple expressions. In addition, the generation of data cubes for supporting user interactions could create large overhead to progressive processing. Because P5 stores the results of progressive processing and the data cubes in different textures for processing with two different WebGL shader programs, the time needed for each progression is at least doubled. For the two WebGL shader programs, a large subset of the computations might be the same since the data cubes include all the visualized data dimensions. Merging these operations properly can avoid duplicated computations, which can remove the overhead to progressive processing due to the computations for generating the data cubes. We plan to address these issues by extending our API and improving our framework for progressive parallel processing in future work. In addition, we plan to extend P5 to include high-level interactions for programmatic selection of features (e.g, hot spots and outliers) from the visualizations of aggregation, clustering, or dimensionality reduction results. Most existing declarative visualization toolkits [7, 35, 36] only provide low-level interaction specification for defining how to make selections by the users. This approach cannot effectively support common visual analysis tasks that make use of machine learning algorithms. For example, selecting all the major clusters to compare their statistics in faceted visualization views requires the users to manually make multiple selections. To address this, high-level interaction specification for describing what to select, instead of how to select, is needed. As we are adding several progressive clustering and dimensionality reduction methods to P5, we also plan to provide declarative grammar with high-level interaction specification for programmatic selection of insights from the results of these data analytics methods. P5 with programmatic selection (P6) will be an even more attractive and powerful toolkit for creating data analysis and visualization solutions.

P5 should be easily adopted by application developers and data analysts given its simplicity and performance advantage. Public deployment and collecting user feedback can help us improve and extend P5. We have set up a work plan for pursuing these.

7 Conclusion

Visualization has become an essential tool in the fast growing areas of data driven discovery, problem solving, storytelling, and learning. The accessibility of existing information visualization software tools helps promote the widespread use of visualization. In many real-world applications, however, the dynamic nature of the data and complex analysis needs demand high performance, progressive processing capabilities that are not generally offered by current toolkits. P5 has been created to meet this demand. We leverage declarative grammars to make progressive visualization and GPU computing more accessible and flexible, allowing more people to use them for building interactive visualization systems. The current implementation of P5, as demonstrated, and the planned extensions promise to support a wide-range of emerging big data applications as well as further increase the impact of visualization as an enabling technology. The source code, documentations, and examples of P5 can be found at `https://github.com/jpkli/pv`.

Acknowledgments

This research is supported in part by the National Science Foundation via grant IIS-1528203 and the Department of Energy via grant DE-SC0014917.

References

[1] Tableau Software. Business Intelligence and Analytics. `https://www.tableau.com/`. Accessed:2019-6-15.

[2] TIBCO Spotfire. Data Visualization and Analytics Software. `https://www.tibco.com/products/tibco-spotfire`. Accessed:2019-6-15.

[3] W3C. File API. `https://www.w3.org/TR/file-upload/`. Accessed:2019-6-15.

[4] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 29–42. ACM, 2013.

[5] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 429–435. ACM, 2002.

[6] S. K. Badam, N. Elmqvist, and J.-D. Fekete. Steering the craft: Ui elements and visualizations for supporting progressive visual analytics. In *Computer Graphics Forum*, vol. 36, pp. 491–502. Wiley Online Library, 2017.

[7] M. Bostock, V. Ogievetsky, and J. Heer. D^3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[8] M. Card. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[9] S. K. Card, A. Newell, and T. P. Moran. The psychology of human-computer interaction. 1983.

[10] C. D. Carothers, D. Bauer, and S. Pearce. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002.

[11] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1082–1090. ACM, 2011.

[12] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. A progressive refinement approach to fast radiosity image generation. *ACM SIGGRAPH computer graphics*, 22(4):75–84, 1988.

[13] K. A. Cook and J. J. Thomas. Illuminating the path: The research and development agenda for visual analytics. Technical report, Pacific Northwest National Laboratory (PNNL), Richland, WA (US), 2005.

[14] F. C. Crow. Summed-area tables for texture mapping. In *ACM SIGGRAPH computer graphics*, vol. 18, pp. 207–212. ACM, 1984.

[15] J.-D. Fekete, D. Fisher, A. Nandi, and M. Sedlmair. Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411). *Dagstuhl Reports*, 8(10):1–40, 2019. doi: 10.4230/DagRep.8.10.1

[16] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[17] D. Fisher, I. Popov, and S. Drucker. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012.

[18] S. Frey, F. Sadlo, K.-L. Ma, and T. Ertl. Interactive progressive visualization with space-time error control. *IEEE transactions on visualization and computer graphics*, 20(12):2397–2406, 2014.

[19] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[20] J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, 2010.

[21] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Acm Sigmod Record*, vol. 26, pp. 171–182. ACM, 1997.

[22] D. Laur and P. Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *ACM SIGGRAPH Computer Graphics*, vol. 25, pp. 285–288. ACM, 1991.

[23] D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang, M. Zu, and W. Chen. Echarts: a declarative framework for rapid construction of web-based visualization. *Visual Informatics*, 2(2):136–146, 2018.

[24] J. K. Li and K.-L. Ma. P4: Portable parallel processing pipelines for interactive information visualization. *IEEE transactions on visualization and computer graphics*, accepted in 2018. (preprint: https://ieeexplore.ieee.org/document/8468065).

[25] J. K. Li, M. Mubarak, R. B. Ross, C. D. Carothers, and K.-L. Ma. Visual analytics techniques for exploring the design space of large-scale high-radix networks. In *IEEE International Conference on Cluster Computing*, pp. 193–203, 2017.

[26] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2456–2465, 2013.

[27] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[28] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, vol. 32, pp. 421–430. Wiley Online Library, 2013.

[29] B. McDonnel and N. Elmqvist. Towards utilizing gpus in information visualization: A model and implementation of image-space operations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1105–1112, 2009.

[30] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE transactions on visualization and computer graphics*, 20(12):1643–1652, 2014.

[31] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):98–108, 2018.

[32] N. Pezzotti, A. Mordvintsev, T. Hollt, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Linear tsne optimization for the web. *arXiv preprint arXiv:1805.10817*, 2018.

[33] H. Piringer, C. Tominski, P. Muigg, and W. Berger. A multi-threading architecture to support interactive visual exploration. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1113–1120, 2009.

[34] D. Ren, B. Lee, and T. Höllerer. Stardust: Accessible and transparent gpu support for information visualization rendering. In *Computer Graphics Forum*, vol. 36, pp. 179–188. Wiley Online Library, 2017.

[35] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017.

[36] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2016.

[37] H.-J. Schulz, M. Angelini, G. Santucci, and H. Schumann. An enhanced visualization process model for incremental visualization. *IEEE transactions on visualization and computer graphics*, 22(7):1830–1842, 2016.

[38] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[39] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel, et al. Tensorflow. js: Machine learning for the web and beyond. *arXiv preprint arXiv:1901.05350*, 2019.

[40] C. D. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014.

[41] C. Stolte, D. Tang, and P. Hanrahan. Multiscale visualization using data cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):176–187, 2003.

[42] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser. Designing progressive and interactive analytics processes for high-dimensional data analysis. *IEEE transactions on visualization and computer graphics*, 23(1):131–140, 2017.

[43] J. R. Wallace, K. A. Elmquist, and E. A. Haines. A ray tracing algorithm for progressive radiosity. In *ACM SIGGRAPH Computer Graphics*, vol. 23, pp. 315–324. ACM, 1989.

[44] H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer, 2016.

[45] L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag, Inc., 1999.

[46] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE transactions on visualization and computer graphics*, 23(8):1977–1987, 2017.