Customizing Computational Methods for Visual Analytics with Big Data

Jaegul Choo and Haesun Park ■ *Georgia Tech*

Computational methods can improve the scalability of visual analytics (VA) by providing compact, meaningful information about the input data. However, the required computation time hinders real-time interactive visualization of big data. By addressing discrepancies between these methods and VA, researchers have proposed ways to customize them for VA.

Owing to the complexities and obscurities in large-scale datasets ("big data"), interest has increased in visual analytics[1,2] (VA) because of humans' ability to quickly gain insight through visual analysis and decision processes. Unfortunately, most state-of-the-art VA tools or techniques don't properly accommodate big data.

The inherent complexities of VA with big data have two main causes. The first is related to human perception. When the number of visualized objects (data items or features) becomes large, humans often have difficulty extracting meaningful information. The second cause is the limited screen space, which often leads to significant visual clutter when a visualization displays too many data items or features.

To improve visual scalability, computational methods can provide compact, meaningful information about the raw data. Such methods include dimension reduction, clustering, and methods that exploit machine learning and data mining. These methods' natural fit and appeal for VA for big data have led to their integration into VA. However, by themselves they don't effectively support real-time interactive VA for big data. Here, we suggest ways to customize them to leverage aspects specific to VA environments.

Hurdles to Full Integration

Achieving appropriate integration faces two main hurdles. First, computational modules and their output can be difficult to understand. Without deep knowledge about the computational methods and data, analysts might find the methods' output more difficult to understand than the original raw data. Many computational algorithms are complicated and often, for the sake of flexibility, involve parameters that must be carefully determined. If the analysts don't understand the parameters' functions, they might improperly set the parameters' values. Thus, many VA systems choose a specific computational method, treat it as a black box, and focus on analysis of its output. However, if the analysts don't properly understand the algorithm and its parameters, a computational module still might not perform well enough to analyze data properly.

Second, computational methods require significant time. Most methods involve heavy computation. As they become more advanced and capable, they tend to require more intensive computation. Such computation usually has a squared or cubic order of complexity in relation to the number of data items or features. So, for big data, the significant computation time required hinders real-time visualization and subsequent interaction with the computational modules.

The most important way over these hurdles is to make computational methods fast enough to ensure real-time visualization and interaction in VA. If computational methods become responsive

 Published by the IEEE Computer Society

in real time, difficulties in understanding them can be partly alleviated by letting users quickly test them in various ways interactively. An initial solution for making them fast could be to parallelize algorithms across multiple cores or CPUs. However, this requires significant effort and time and creates significant communication overhead. Furthermore, significant portions of some algorithms can't be parallelized. So, although this technique might achieve a minor amount of speedup, real-time VA demands much more.

Exploiting Discrepancies

To provide the necessary speed, we exploit several crucial discrepancies between computational methods and VA. To discuss these discrepancies, we focus on the two aspects of VA we mentioned earlier: human perception and screen space. Many computational methods don't consider these aspects because those methods aren't entirely intended for VA. So, they tend to compute excessively precise results compared to what humans can perceive in VA. If we could remove excessive computation, computational methods would become much faster and allow for real-time interaction while providing the human eye with comparable visual results. Here, we examine this excessive computation in more detail from two angles: precision and convergence.

Precision

Most computational methods work with the precision of modern CPUs, which is typically double precision. This precision gives at least 10 significant decimal digits, and usually gives 15 to 17 digits.

Perception-based precision. Such precision, however, might be superfluous for humans' perception capability. For example, most people know that π is approximately 3.14. Although some people might know a more accurate value—for example, 3.141592—many wouldn't usually care much about the exact value.

Or, suppose that a document-topic-modeling algorithm gives a certain document's topical weight vector as (17.3952%, 78.1541%, 4.4507%) for three topics: sports, science, and politics. People will likely perceive a topic's weight at a precision of tenths at most (17.4%, 78.2%, 4.5%). However, having more accurate numbers won't change their perception significantly.

Screen-space-wise precision. The precision in VA is also bound by the limited resolution of the screen space displaying the computational results. That is, even though the results are represented in fine-grained numbers, they're quantized into a much coarser grain owing to the limited resolution.

For example, suppose a force-directed graph layout method visualizes data in a scatterplot. The visualization converts the resulting 2D coordinates to the data points' positions on a screen with a limited number of pixels. For instance, if the screen resolution is 1,024 × 768, each 2D coordinate is quantized and mapped to 1,024 and 768 different values, respectively. In most cases, such a granularity is coarser than what the computational algorithm used—for example, single or double precision. Therefore, even if the graph layout computes a 2D coordinate of (1.34643678, 3.02862473), this might be converted to (1.35, 3.03) on the screen. In this sense, computational algorithms performed at the computers' precision level are excessively accurate for visualization.

Many computational algorithms are complicated and often, for the sake of flexibility, involve parameters that must be carefully determined.

Convergence

Computational methods have become so complicated that they often have no closed-form solution. Instead, many of them iteratively refine the solution until it converges to a final solution. The notion of convergence becomes critical in determining when to terminate the iterations, and algorithms have their own stopping criteria based on many well-studied theories and principles.

Perception-based convergence. For many computational methods, one important characteristic of convergence is that the major refinement of the solution typically occurs in early iterations, whereas only minor changes occur in later iterations. Considering that humans can quickly understand the overall structure or trend in data, they might be able to perceive enough information from an intermediate result obtained from an earlier iteration.

For example, many clustering algorithms iteratively refine the cluster membership or coefficient for each data item. More specifically, given initial cluster memberships of data items, *k*-means clustering,[3] a widely used algorithm, alternates two steps iteratively:

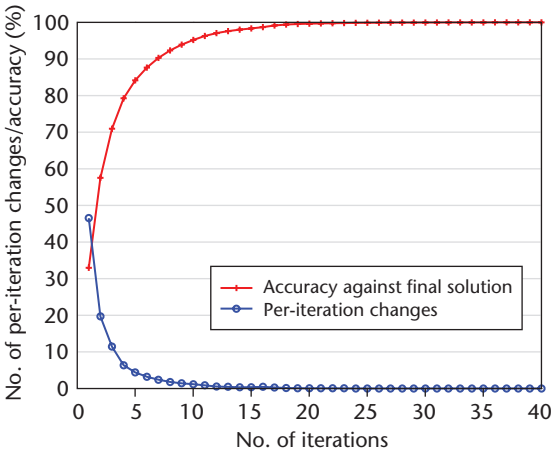


Figure 1. The relative changes of cluster memberships between iterations, and the cluster membership accuracy with respect to the final converged solution. This example used *k*-means clustering to cluster 50,000 Reuter newswire articles into 20 clusters. Most of the time for running the algorithm could be curtailed because it doesn't contribute much to a human's perception in visual analytics (VA).

1. Compute each cluster's centroid by averaging the feature vectors of that cluster's data items.
2. Update each data item's cluster assignment on the basis of its closest cluster centroid.

The iteration terminates when no membership changes occur.

For instance, we used *k*-means clustering to cluster 50,000 Reuter newswire articles into 20 clusters. Figure 1 shows how many cluster membership changes occurred throughout the 40 iterations. Major changes occurred in only the first few iterations. For instance, fewer than 5 percent of the data items changed their memberships after the fifth iteration, as the blue line shows. After the seventh iteration, more than 90 percent of the data items had been correctly clustered, as the red line shows. In addition, each iteration of the *k*-means algorithm requires an equal amount of time. Therefore, most of the time for running the algorithm could be curtailed because it doesn't contribute much to a human's perception in VA.

Screen-space-wise convergence. The coarse-grained quantization due to the screen space's limited resolution can also affect convergence. To better describe the idea, we show an example using multidimensional scaling (MDS), a common computational method. Basically, MDS tries to preserve all the pairwise distances or relationships of data items in the lower-dimensional space, which is typically a 2D or 3D screen space.

In particular, our example uses nonmetric MDS, which tries to preserve the distance values' orderings instead of their actual values. Nonmetric MDS is often better suited to VA than the original MDS because humans care more about data items' ordering. However, it requires much more intensive computation than metric MDS.

We used nonmetric MDS on 2,000 data items consisting of handwritten numbers. Of the 169 iterations, major changes occurred in only the first few (see Figure 2a). After the fourth iteration, a data item's average pixel-wise coordinate change was fewer than 10 pixels from the item's coordinates in the previous iteration, as the blue line shows. After the 30th iteration, each data item was on average fewer than 10 pixels away from the final converged coordinate, as the red line shows. Scatterplots generated by the fifth iteration and the converged result (see Figures 2b and 2c, respectively) confirm that the changes between the two are indeed minor.

Customizing Computational Methods

Here, we suggest how to customize computational methods by tackling the precision and convergence discrepancies.

Low-Precision Computation

One of the easiest ways to lower precision and thus accelerate computation is to change double precision to single precision.

Figure 3 shows the results of using principal component analysis (PCA)[4] to generate a scatterplot of facial-image data with single and double precision. Single precision took much less time, but the two cases generated almost identical scatterplots. After analyzing the exact pixel-wise coordinates at 1,024 × 768 resolution, we found only two pixel-wise displacements between the two cases.

You could more carefully determine the computational precision on the basis of human perception and the screen resolution. For instance, you could conduct a user study on how significantly a human's perception of the results degrades as the precision decreases. Conversely, you could formulate the minimum precision required for a given resolution of the screen space.

So far, little research has focused on adopting a lower precision than the standard double precision to save computation time. Researchers have studied computation at an arbitrary precision, but their primary purpose was to support much higher precision than modern CPUs can handle.[5]

However, considerably decreasing the precision might not always achieve computational efficiency owing to hardware issues. Specifically, most CPUs have a floating-point unit (FPU)—a dedicated

coprocessor that specializes in double-precision floating-point operations. Therefore, computation with an arbitrarily low precision can't easily benefit from FPUs. Nonetheless, in embedded systems in mobile devices, which typically use low-cost microprocessors without an FPU, lowering the precision can significantly boost interactive visualization applications' performance.

In addition, such low-precision computation can turn most computation into fixed-point arithmetic, which essentially represents a number as an integer scaled by a specific binary or decimal factor. Although fixed-point arithmetic could break down the numerical stability, embedded systems have actively used it for efficient computation. Extending this idea to VA could accelerate many computational methods while maintaining their visual results' quality when numerical stability isn't a critical concern.

Iteration-Level Interactive Visualization

As we discussed before, an algorithm can reach a viable solution for VA much earlier than its converged iteration. However, how to determine such convergence criteria still isn't clear. Instead, we propose *iteration-level interactive visualization*, which aims at visualizing intermediate results at various iterations and letting users interact with those results in real time.

As Figure 4 shows, this approach exploits a useful characteristic of many iterative algorithms—the intermediate result at each iteration shares the same form as the final output. For instance, many clustering algorithms, including *k*-means clustering (see Figure 1), give the cluster memberships or coefficients of the entire set of data items at each iteration and become more refined as iterations proceed. In other words, you don't just obtain the clustering result for the first item at the first iteration, the result for the second item at the next iteration, and so on. Therefore, visualizing the intermediate results can provide information about the entire data, although certain individual data items might not be as precise in earlier iterations.

Because this approach dynamically visualizes the intermediate results, users can quickly get an overview of them. Furthermore, interaction with computational methods can become much more responsive because interactions can be reflected in later iterations. So, if users make a specific change, such as a parameter change, to a computational method, they won't need to wait through a complete run of iterations to see the change's results.

Iteration-level visualization isn't a completely new idea. For instance, force-directed graphs implemented

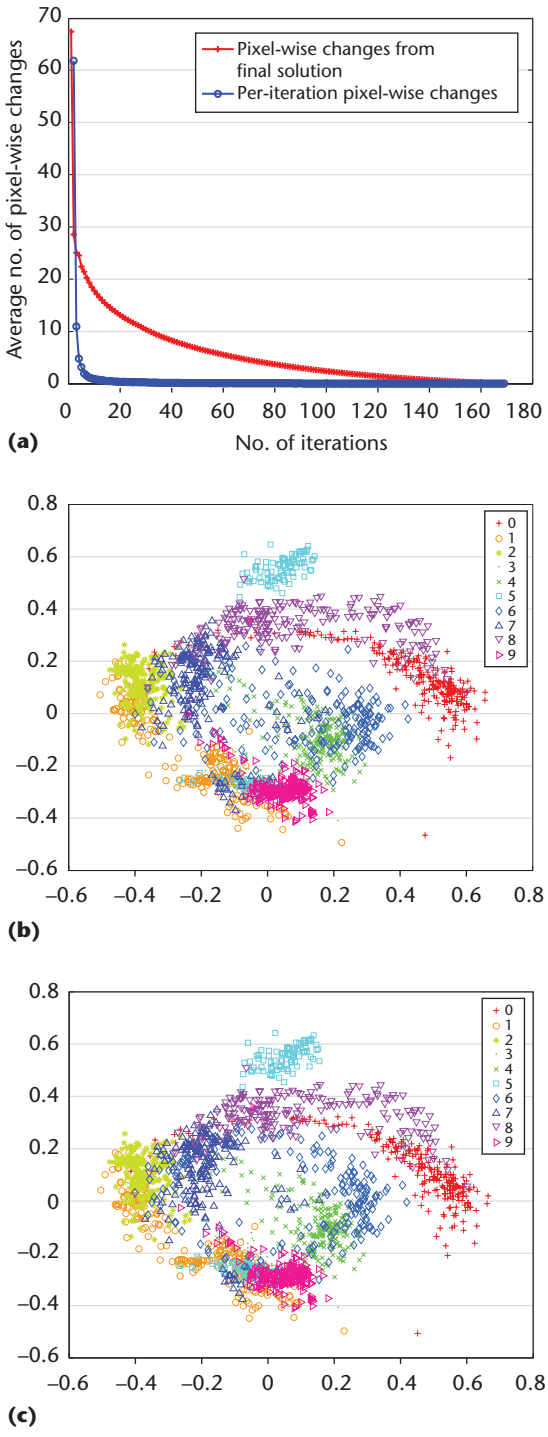


Figure 2. A scatterplot using nonmetric multidimensional scaling. (a) The averaged sum of the horizontal and vertical pixel position differences for each data item between iterations, and the same measures with respect to the final converged solution. (b) The scatterplot generated by the fifth iteration. (c) The scatterplot generated by the converged result (the 169th iteration). In this example, we used 2,000 data items consisting of handwritten numbers and color-coded the number labels. The changes between Figures 2b and 2c are minor.

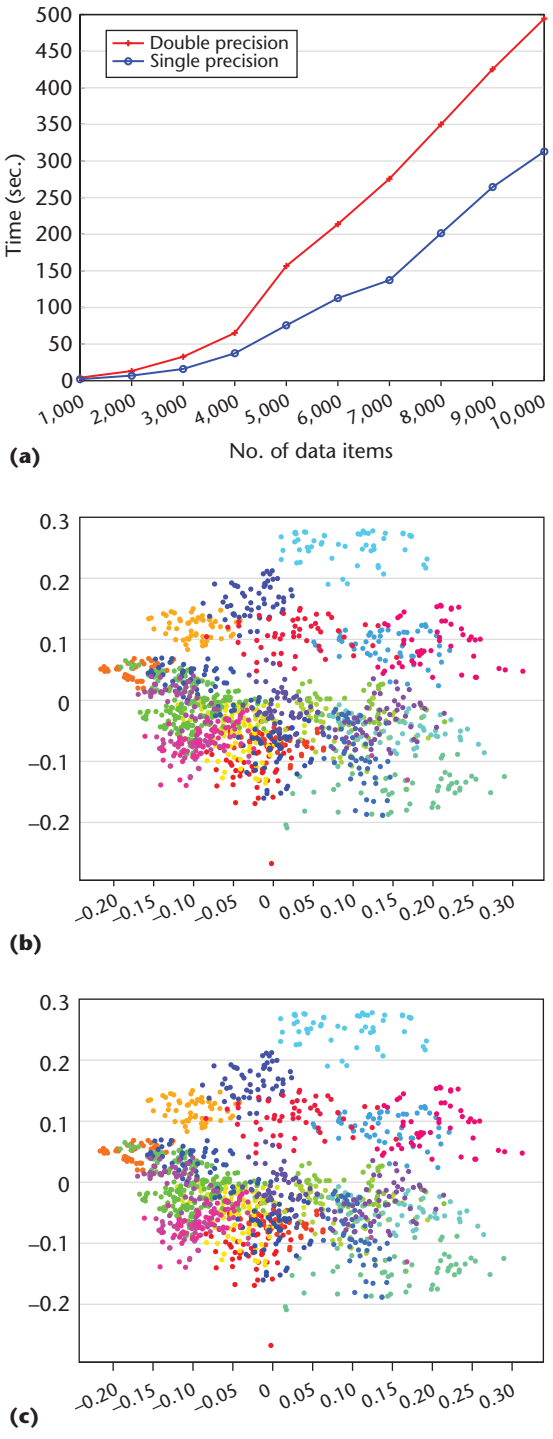


Figure 3. Using principal component analysis to generate scatterplots with single and double precision. (a) The computation time depending on the data size. (b) The scatterplot based on single-precision computation. (c) The scatterplot based on double-precision computation. This example used facial-image data. For Figures 3b and 3c, we visualized 1,420 data items and color-coded them with their person ID. We represented each data item as an 11,264-dimensional vector. We found only two pixel-wise displacements between the two cases.

with the D3 (Data-Driven Documents; http://d3js.org) or Prefuse (http://prefuse.org) toolkits visualize data immediately after the first iteration. Nonetheless, we claim that researchers have overlooked the fact that many computational methods are iterative.

This neglect is partly because customizing computational methods sometimes requires knowledge of how they work. For example, people unfamiliar with eigendecomposition, PCA's main algorithm, might consider PCA to be a black-box algorithm. Actually, eigendecomposition is an iterative algorithm consisting of, say, the power iteration, Lanczos iteration, and so on. So, you can redesign PCA for iteration-level visualization. Many other iterative algorithms could also be customized for real-time VA.

Iterative Refinement of Computational Results

Some user interactions might require more precise information than what we've been describing.

One obvious example is zooming in or out. Suppose the precision and convergence in the algorithm that generated a scatterplot have been adjusted on the basis of the screen's resolution. If the user zooms in on a specific area, the 2D coordinate information of data items in that area must be represented with a finer granularity.

Unlike the traditional approaches that give fully precise results, our approaches require further computation to obtain more precise results to support zoom-in and zoom-out on demand. To improve this situation, we could iteratively refine the computational results to a higher precision. Figure 5 shows such an example. During refinement, each step employed the previous step's results, which made each step efficient. To determine a specific data item's position at a higher resolution, the refinement needed to examine only the nearby areas from the previous position.

Research in other domains points to possible ways to realize this approach. Example techniques include adaptive mesh refinement[6] in numerical analysis and wavelet transform[7] in image coding or compression. Applying these ideas to integrate computational methods in VA would open up an interesting new research area.

Data Scale Confinement

As Figure 5a clearly shows, the screen space's resolution limits the number of data items that can be visualized. Suppose there are many more data items than available pixels. Processing the entire dataset doesn't make much sense because there's no way to visualize all the items.

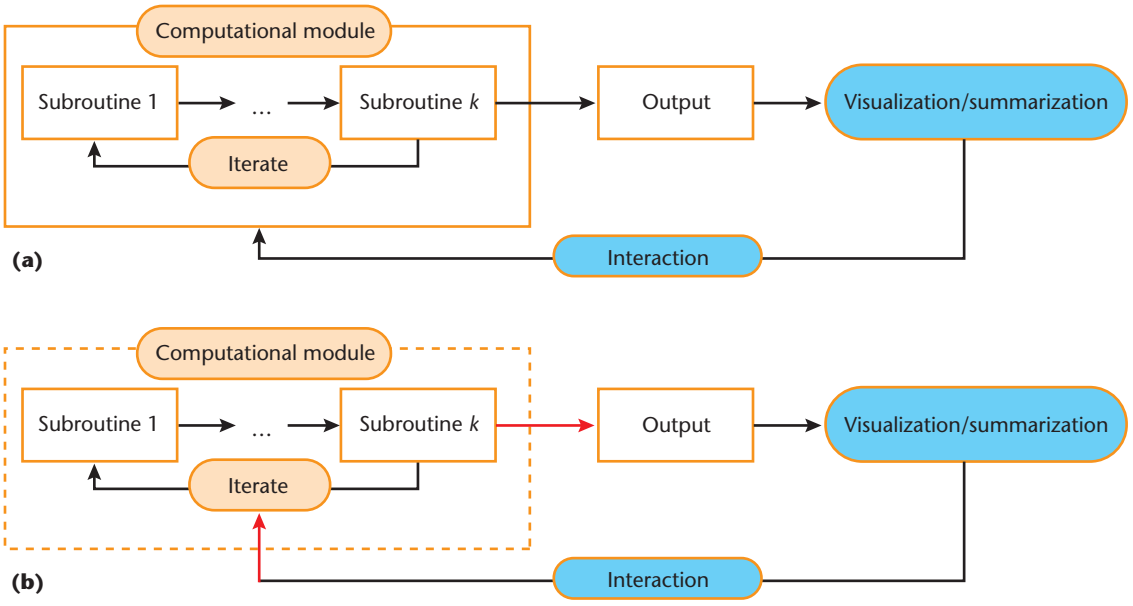


Figure 4. Two approaches to applying computational methods to VA. (a) In the standard approach, visualization and interaction occur only after the computational module finishes its iterations. (b) In iteration-level interactive visualization, intermediate results are visualized dynamically; users can interact with the computational module during iterations.

Data scale confinement is particularly useful for dealing with computational complexity. In principle, as the number of data items increases, the algorithm complexity can't be more efficient than $O(n)$, which assumes that every data item is processed at least once. Even such ideal complexity can cause a computational bottleneck in real-time VA. Having a fixed number of available pixels can turn algorithmic complexity into $O(1)$, in that you can visualize only a specific number of data items at most. One of the easiest ways to select this data subset is random sampling, although you could adopt other more carefully designed sampling methods that better represent the entire dataset.

Some user interaction such as zoom-in or zoom-out might require the computational results for data items that haven't yet been processed. In this case, you can handle the situation through a different kind of efficient computation.

For example, suppose you have a large-scale dataset for which only a certain subset of the data has been clustered. To obtain the remaining data items' cluster labels, you can apply a simple classification method based on the already computed clusters.

Or, in the case of dimension reduction, suppose PCA has been computed on a data subset. You can project the remaining data onto the same space via a linear transformation matrix given by PCA. This is much more efficient than computing PCA on the entire dataset.

Although these approximated approaches can't give the exact same results as those generated by using the entire data from the beginning, they're a viable way to ensure real-time VA for big data.

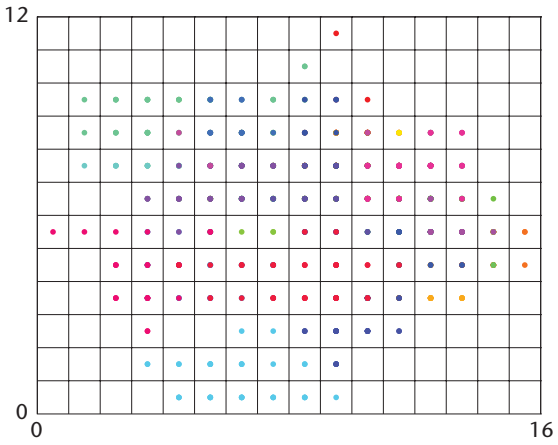
To achieve tight integration between computational methods and VA, researchers from each side must care more about the other side. In particular, researchers who design computational methods must realize that making an algorithm more interactive and interpretable in practical data analysis scenarios is just as important as addressing practical concerns such as the data's maximum applicable size, computation time, and memory requirements. On the other side, researchers who apply computational methods to VA need to understand the algorithm details as much as possible and tailor them to make them blend well in real-time VA.

Acknowledgments

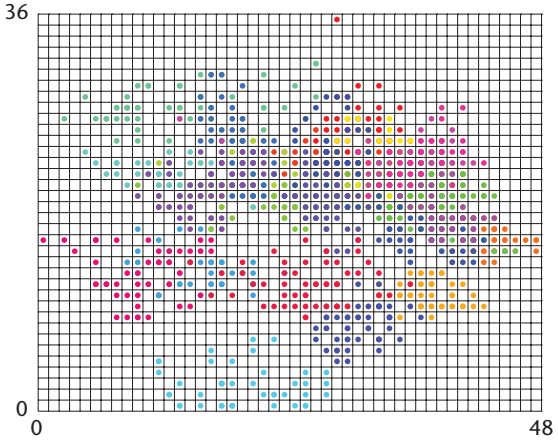
US National Science Foundation (NSF) grant CCF-0808863 partly supported this research. Any opinions, findings, and conclusions or recommendations in this article are the authors' and don't necessarily reflect the NSF's views.

References

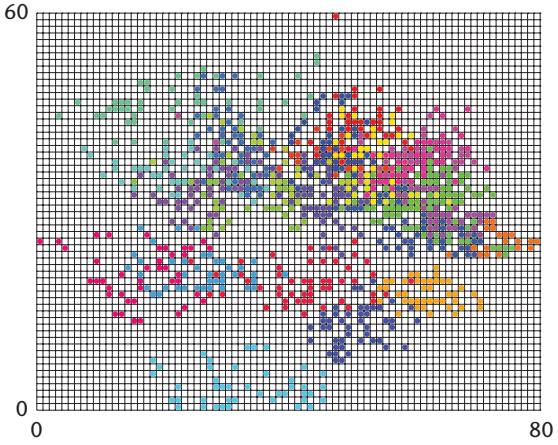
1. D. Keim, "Information Visualization and Visual Data Mining," *IEEE Trans. Visualization and Computer Graphics*, vol. 8, no. 1, 2002, pp. 1–8.
2. J. Thomas and K. Cook, *Illuminating the Path: The*



(a)



(b)



(c)

Figure 5. Hierarchical precision refinement of computational results for a scatterplot at (a) 16 × 12, (b) 48 × 36, and (c) 80 × 60 resolution. This figure uses the same data and method as in Figure 3. To determine a specific data item's position at a higher resolution, the refinement needed to examine only the nearby areas from the previous position.

Research and Development Agenda for Visual Analytics, IEEE CS, 2005.
3. C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
4. I.T. Jolliffe, *Principal Component Analysis*, Springer, 1986.
5. D. Knuth, *The Art of Computer Programming*, Addison-Wesley, 2006.
6. M. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *J. Computational Physics*, vol. 82, no. 1, 1989, pp. 64–84.
7. C.K. Chui, *An Introduction to Wavelets*, Academic Press, 1992.

Jaegul Choo *is a research scientist and PhD candidate in computational science and engineering at Georgia Tech. His research interests include visualization, data mining, and machine learning, focusing on dimension reduction and clustering methods. Choo received an MS in electrical engineering from Georgia Tech. Contact him at jaegul.choo@cc.gatech.edu.*

Haesun Park *is a professor and an associate chair in Georgia Tech's School of Computational Science and Engineering. She's also the director of the Foundations of Data and Visual Analytics center and executive director of the Center for Data Analytics, both at Georgia Tech. Her research interests include numerical algorithms, data analysis, visual analytics, text mining, and parallel computing. Park received a PhD in computer science from Cornell University. She's a Society for Industrial and Applied Mathematics Fellow. Contact her at hpark@cc.gatech.edu.*