Attribute Preserving Dataset Simplification

Jason D. Walter and Christopher G. Healey*
Department of Computer Science, North Carolina State University

Abstract

This paper describes a novel application of feature preserving mesh simplification to the problem of managing large, multidimensional datasets during scientific visualization. To allow this, we view a scientific dataset as a triangulated mesh of data elements, where the attributes embedded in each element form a set of properties arrayed across the surface of the mesh. Existing simplification techniques were not designed to address the high dimensionality that exists in these types of datasets. As well, vertex operations that relocate, insert, or remove data elements may need to be modified or restricted. Principal component analysis provides an algorithm-independent method for compressing a dataset's dimensionality during simplification. Vertex locking forces certain data elements maintain their spatial locations; this technique is also used to guarantee a minimum density in the simplified dataset. The result is a visualization that significantly reduces the number of data elements to display, while at the same time ensuring that high-variance regions of potential interest remain intact. We apply our techniques to a number of well-known feature preserving algorithms, and demonstrate their applicability in a real-world context by simplifying a multidimensional weather dataset. Our results show a significant improvement in execution time with only a small reduction in accuracy; even when the dataset was simplified to 10% of its original size, average per attribute error was less than 1%.

CR Categories: I.3.6 [Computer Graphics]: Computational Geometry and Object Modeling—curve, surface, solid, and object representation; geometric algorithms, languages, and systems

Keywords: dataset management, mesh simplification, principal component analysis, scientific visualization

1 Introduction

Much of the work in scientific visualization deals with the intelligent management and display of large, complex collections of data. Datasets D containing many millions of elements e_i are not uncommon, moreover, each element may encode multiple attribute values A_j (*e.g.*, a weather dataset where each element represents weather station readings for $n = 9$ attributes: latitude, longitude, elevation, time, temperature, pressure, wind speed, humidity, and precipitation). Our goal during visualization is to convert some or all of these values into images that allow viewers to *explore*, *analyze*, *verify*, and *discover*.

The problems of dataset size and dimensionality were identified and discussed by the original NSF panel on scientific visualization [9]. Although significant advances have been made in recent years (*e.g.*, with methods like spot noise and line integral convolution, perceptual visualization, and feature extraction and data mining) [10], even the most sophisticated techniques are often unable to display a dataset in its entirety [12]. A dramatic increase in our ability to collect and archive enormous amounts of raw data have further emphasized the need to study the problem of dataset management.

Processing large sets of information for on-screen display is not unique to visualization. One area of particular interest is the control of geometry during rendering in computer graphics. Researchers are studying how to generate mesh representations of 3D objects that are highly detailed, yet small enough in their polygon count to render at interactive frame rates. Initial work in this area proposed a number of simplification techniques that characterize different parts of an object by their level of geometric detail. The resolution of the mesh is varied to be expressive in areas of high detail (*i.e.*, high-resolution, with many small triangles) and compact in areas of low detail (*i.e.*, low-resolution, with only a few large triangles). Recent work has extended these techniques to consider surface properties together with the object's geometry. These *feature preserving mesh simplification* algorithms maintain a high-resolution mesh in locations with sharp variations in geometry or in surface details like color and texture. The result is a model with a significant reduction in polygon count, yet with a visual appearance that is often indistinguishable from the original, full-resolution mesh.

Our interest in feature preserving mesh simplification stems from the belief that a 3D object and a multidimensional dataset have a number of important parallels. The spatial coordinates used to position a data element during visualization form a set of vertices that can be triangulated to produce an underlying mesh. The individual attributes associated with each element can then be viewed as a set of properties arrayed across the surface of the mesh. Conceptualizing the dataset in this manner suggests we may be able to apply feature preserving simplification algorithms to reduce its size (*i.e.*, the polygon and associated vertex count) based on attribute variability. Spatial regions with near-constant attribute values could be reduced to only a few elements; areas with high levels of variation would be densely populated. In addition to the absolute reduction in dataset size, knowing *where* simplification occurs would provide a viewer with valuable information about when attributes vary and how they interact with one another.

Unfortunately, most feature preserving simplification algorithms cannot be applied verbatim to a multidimensional dataset. These techniques were designed for a 3D modeling environment, and were not meant to be used for reducing the size of a high-dimensional data collection. This can result in unanticipated problems, for example:

- Many feature preserving techniques were optimized to handle only a few surface properties (*e.g.*, (r, g, b) color, or texture (u, v) coordinates); performance and accuracy can degrade significantly in the presence of the tens or hundreds of attributes that exist in many multidimensional datasets.
- Algorithm-specific methods for error estimation, vertex elimination, and vertex relocation are employed; this necessitates "respecifying" a dataset (often in non-trivial ways) to fit each algorithm's mesh and surface property assumptions.
- Certain vertex operations may not make sense in a multidimensional visualization context; since vertices represent data elements, it may not be allowable to arbitrarily add, remove, or relocate them.

Our goal is a method that allows us to harness any of the current (or future) simplification techniques without the need for extensive

*Department of Computer Science, 1010 Main Campus Drive, North Carolina State University, Raleigh, NC, 27695-7534; healey@csc.ncsu.edu

modifications to fit the algorithm, and with the expectation of an accurate, high-quality result. In order to do this, we must address a number of relevant problems:

1. Design algorithm-independent methods for allowing rapid and accurate error estimation and vertex management for high-dimensional datasets.
2. Develop algorithm-independent techniques to protect specific data elements from certain types of modifications.
3. Ensure our extensions integrate seamlessly into existing feature preserving simplification algorithms.

The remainder of this paper describes our solutions to these problems. Section 2 provides an overview of related work in mesh simplification and dataset management in visualization. Section 3 begins with an introduction to principal component analysis, followed by a description of its use for rapid and robust error estimation and vertex management. Section 4 explains how our new techniques were applied to existing feature preserving simplification algorithms. Section 5 demonstrates their application to a real-world dataset of environmental weather readings. Section 6 concludes with a summary of our results.

2 Related Work

We discuss briefly previous research in two areas most closely related to our own studies: (1) intelligent management of large, complex datasets in scientific visualization, and (2) feature preserving mesh simplification techniques from computer graphics.

2.1 Dataset Management

Although management of scientific datasets has been cited as an important area of current and future research [12], work to date in the visualization community has been limited. Initial studies centered around the use of a common data format that would feed through data filters and on to high-level visualization tools [14]. Although this is an important consideration, it does not address *how* to design filters and visualization techniques to compress and display a dataset in an intelligent manner. Some recent systems were built on top of a relational database, thereby harnessing its power to perform data organization and SQL operations (*e.g.,* in [13]). Unfortunately, certain properties common to scientific datasets like errors, noise, duplicate records, and missing values make them difficult to integrate into a relational data model. Feature extraction has been applied in certain cases to automatically identify and isolate regions of interest in a large dataset (*e.g.,* in [5, 11]); only these extracted regions are shown in the final visualization.

2.2 Mesh Simplification

Numerous techniques have been proposed to perform geometric simplification on an underlying 3D mesh. Less work has studied the problem of preserving geometry and surface properties together during simplification. We focus our discussion of related work on these feature preserving algorithms. In fact, many of these methods are extensions of previous techniques that considered geometric error alone.

Bajaj and Schikore [1] apply vertex removal and reprojection to simplify a 3D mesh with surface properties; the goal of this algorithm is closest to our own: the simplification of a surface representing multivariate data. Unfortunately, this technique does not address performance and accuracy issues that can occur during the unrestricted simplification of a high dimensional dataset. Hoppe [7] defines a progressive mesh structure, a series of edge collapses that produce a monotonic reduction in mesh complexity; mesh geometry and scalar and discrete surface properties are preserved by minimizing an energy function that measures the deviation between the simplified and the original mesh. Garland and Heckbert [3] contract pairs of vertices to simplify a model; the error introduced during each operation is estimated using quadrics. Surface properties are treated as extensions to each vertex definition [4], and to the error matrices used to process them. Hoppe [8] proposed a redefinition of the quadric error matrix used in [4] to reduce storage cost and improve accuracy; the new error metric separates geometric error (based on the distance a simplified vertex v_i' strays from its projected position v_i on the original mesh face) and surface property error (based on the deviation between properties assigned to v_i' and the actual property values stored on the mesh face at v_i) to improve both efficiency and visual appearance. Cohen et al. [2] separate a model into a geometric surface and one or more surface maps (*e.g.,* a texture map and a normal map). Simplification envelopes are used to identify a sequence of operations to reduce the model's geometry; operations are ordered based on the the amount of deviation they produce in each surface map.

3 Error and Vertex Management

Existing simplification algorithms specify mesh geometry in a consistent fashion as a connected set of vertices in $\mathbb{R}^3$. Because of this, it was not necessary to change how the basic geometry was maintained. Our focus is on managing the surface properties that sit on top of the underlying mesh. We seek a method to compress an n-dimensional dataset into p new dimensions, $p \ll n$, while at the same time ensuring this reduction does not remove important details of interest to a viewer. We chose to use principal component analysis to address this problem.

3.1 Principal Component Analysis

Principal component analysis (PCA) forms linear combinations of the original attributes $A_1, \ldots, A_n$ to construct a sequence of principal component axes $Z_1, \ldots, Z_n$ that span the n-dimensional space containing the data elements. The direction of each Z_i is chosen to maximize its capture of the variance σ^2 contained in a dataset D. The axes are ordered such that $\sigma^2(Z_1) \geq \ldots \geq \sigma^2(Z_n)$. In practice, it often takes only a very few axes p, $p \ll n$, to account for the majority of the variance that exists in a dataset. A viewer-specified cutoff τ is used to select the first p axes such that $\sigma^2(Z_1) + \cdots + \sigma^2(Z_p) \geq \tau$ (*i.e.,* the smallest set of axes that capture at least τ of the variance in D).

Principal component analysis raises an interesting question: "Why not simply transform a dataset to the first p principal component axes (thereby reducing the dimensionality from n to p, yet capturing τ of the variance), then visualize it?" Unfortunately, visualizing data in principal component space does not work well in practice. In most cases viewers are unable to mentally transform the data elements they see back to their original n attribute values. We *can* perform error estimation and vertex management in principal component space, however. This requires little or no change to the existing simplification techniques; from an algorithm's perspective, the only difference is a significant reduction in the number of surface properties to consider. Viewers select τ to guarantee that the p new dimensions capture an appropriate percentage of the variance that exists in D. Results in principal component space are transformed back to the n-dimensional attribute space prior to visualization. This allows viewers to see the simplified dataset in its original, expected context.

Principle component axes are constructed using eigenvalues and eigenvectors. Given an $n \times n$ real, symmetric matrix A of rank n,

there exist scalars λ_i and vectors Z_i such that:

$$\left.\begin{array}{rcl} AZ_i & = & \lambda_i Z_i, \\ (A-\lambda_i I)Z_i & = & 0 \end{array}\right\} \; i = 1, \ldots, n \qquad (1)$$

The non-zero λ_i and Z_i are the *eigenvalues* and *eigenvectors* of A. The determinant $|A - \lambda I|$ is used to solve $|A - \lambda I| = 0$; the n solutions of λ for this polynomial equation are normally ordered and assigned such that $\lambda_1 \geq \ldots \geq \lambda_n$. The λ_i are then used in Eq. 1 to solve for the mutually orthogonal Z_i. Given $Z = [Z_1 \cdots Z_n]$ with each Z_i normalized, it follows that $Z^T Z = I$, or $Z^T = Z^{-1}$. We can use this result to rewrite Eq. 1 as:

$$\begin{array}{rcl} AZ & = & ZL, \\ A & = & ZLZ^T \end{array} \qquad (2)$$

where L is the diagonal matrix of eigenvalues.

To solve for principal component axes, a dataset D is rewritten as an $m \times n$ matrix of m data elements over n attributes. Individual attributes are normalized to allow for relative comparisons. The $n \times n$ covariance matrix C of the dataset D is then used to measure the variance between all pairs of attributes i and $j, 1 \leq i, j \leq n$. Specifically:

$$\begin{array}{rcl} C_{i,j} & = & E(XY) - E(X)E(Y) \\ & = & \dfrac{\sum_{k=1}^{n} D_{k,i} D_{k,j}}{n-1} - \dfrac{\sum_{k=1}^{n} D_{k,i} \sum_{k=1}^{n} D_{k,j}}{n} \end{array} \qquad (3)$$

The eigenvectors Z_i of C form the principal component axes of D. The eigenvalues λ_i measure the amount of variance each Z_i captures:

$$\sigma^2(Z_i) \quad = \quad \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \qquad (4)$$

Thus, the first p axes that satisfy $(\lambda_1 + \cdots + \lambda_p) / \sum_{j=1}^{n} \lambda_j \geq \tau$ form a p-dimensional space that captures τ of the variance in D.

3.2 Reconstruction to Attribute Space

Vertex operations made by the simplification algorithms in principal component space represent the relocation of data elements in attribute space. Appropriate attribute values for these relocated data elements must be selected during the transformation back to the original attribute space prior to visualization.

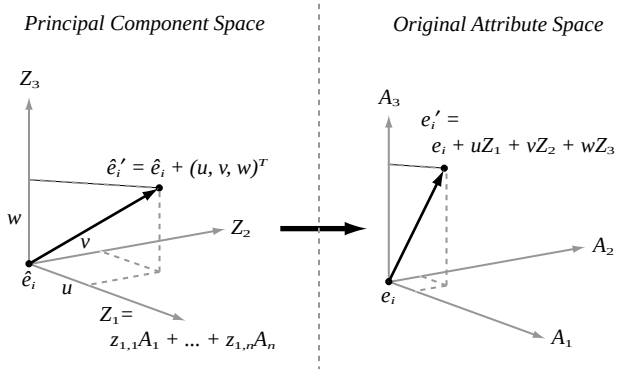


Figure 1: An example of converting a point e_i' translated (u, v, w) in principal component space back to n-dimensional attribute space

Consider a data element e_i represented as $\hat{e_i}$ in principal component space; its attribute values $(a_{i,1}, \ldots, a_{i,n})$ form an anchor from p-dimensional principal component space back to n-dimensional attribute space. After being translated by (u, v, w), the modified data element $\hat{e_i}'$ is reconstructed as e_i' in attribute space via:

$$e_i' \quad = \quad e_i + uZ_1 + vZ_2 + wZ_3 \qquad (5)$$

where Z_j is the j-th eigenvector stored as the j-th column in Z. Intuitively, the coefficients in Z_j are used to estimate the change for each attribute A_j from its anchor point $a_{i,j}$. This follows from Eq. 2, which states:

$$\begin{array}{rcl} \hat{e_i} & = & Z^T e_i \\ \hat{e_i}' & = & Z^T e_i' \end{array} \qquad (6)$$

This can be rewritten as:

$$\hat{e_i}' - \hat{e_i} \quad = \quad Z^T (e_i' - e_i) \qquad (7)$$

If $n = p$ then $Z^T = Z^{-1}$ (since Z is orthonormal), therefore:

$$Z(\hat{e_i}' - \hat{e_i}) + e_i \quad = \quad e_i' \qquad (8)$$

exactly as shown in Eq. 5. If $p < n$ then $Z^T \neq Z^{-1}$, and results from Eq. 7 represent estimates based on the amount of variability captured by Z. Since we expect $p \ll n$, a small amount of error is normally unavoidable. The p axes we select will not capture all of the variance contained in D, so they cannot be used to perfectly reconstruct e_i' in attribute space.

Although reconstruction errors cannot be eliminated, they can be controlled through the use of τ. Large errors imply high spatial frequency features that were missed during reconstruction. However, such a feature should have been captured in one of the principal component axes for any reasonable value of τ. The dependence on τ provides an intuitive method for controlling the tradeoff between speed and accuracy during simplification. From a viewer's perspective, increasing τ produces the expected result: the amount of variance captured in principal component space increases, providing a corresponding increase in accuracy during simplification, but at a potential cost in speed from an increase in the number of principal component axes (or surface properties) that must be considered during simplification.

3.3 Vertex Locking

Each of the simplification algorithms apply some type of vertex relocation, insertion, and removal. This makes sense for a 3D object, since maintaining the starting position of a vertex is normally unimportant. This is not necessarily true in a visualization environment. We must ensure that vertex operations do not introduce certain types of artifacts in the simplified dataset, for example:

1. *Relocation Errors:* Consider a weather dataset where some elements represent weather stations, and others represent interpolated values used to fill locations with no actual readings. The highly accurate weather station elements could be locked to prevent their relocation or removal (while still allowing the interpolated elements to be modified during simplification).
2. *Planing:* In the absence of any preference for particular elements, viewers should lock a structured subset of vertices to guarantee a minimum element density in the simplified dataset. This is important; without a minimum density large holes will appear in areas of near-constant variation, leaving no indication of the type of data that existed in the area prior to simplification.

Although most algorithms provide methods for addressing discontinuities or creases in a mesh, we wanted to build a simple, efficient locking scheme that is algorithm-independent. To this end, we attach a lock flag to each vertex. This flag is checked during each vertex operation; if the flag is set, certain modifications are constrained

or disallowed entirely. We chose to monitor edge collapses; overseeing this single operation has proven sufficient to guarantee the protection of locked vertices for the algorithms we have extended (this choice was motivated by Hoppe [7], who suggested that edge collapses alone can produce effective mesh simplifications; the algorithms we considered used edge collapses directly [2, 4], or used vertex operations that can be easily respecified as an edge collapse [1]). If one endpoint is locked, the edge must collapse to that endpoint. If both endpoints are locked, the edge collapse is not allowed (note that this situation is rare; in most cases locked vertices are located far from one another, so having both endpoints locked implies a very long edge is being considered for collapse).

4 Extended Simplification Algorithms

In order to study our extensions, we integrated them into three existing simplification algorithms: progressive meshes [7], quadrics with integrated geometry [4], and quadrics with separated geometry [8]. The speed and accuracy of the original and extended versions of each algorithm were then tested in the context of a large, multidimensional weather dataset.

4.1 Progressive Meshes

The progressive mesh representation described by Hoppe [7] simplifies a full-resolution mesh M^m with m vertices into progressively coarser meshes $M^{m-1}, \ldots, M^1, M^0$ via a sequence of edge collapses. The geometry of the mesh is stored as a tuple (K, V), where K is a simplicial complex defining connectivity in the mesh, and V is the set of mesh vertices positioned in $\mathbb{R}^3$. A topological realization $|K| \subset \mathbb{R}^m$ is built by associating vertices in K with basis vectors in $\mathbb{R}^m$. The function $\phi_V(|K|) : \mathbb{R}^m \rightarrow \mathbb{R}^3$ maps the mesh back to 3D space. Discrete surface properties $d_f \in D$ are associated with a face $f = \{j, k, l\} \in K$, while scalar properties $s_{(v,f)} \in S$ are associated with the corners $(v, f) \in K$.

An edge collapse transforms two vertices (v_i, v_j) into a single vertex v_i'. Vertex v_j and its two adjacent faces are removed from the mesh. The order of edge collapses is carefully chosen to minimize the geometric and surface property errors they introduce. This goal is redefined as an energy minimization problem:

$$E(M^i) = E_{geom}(M^i) + E_{scalar}(M^i) + E_{disc}(M^i) \quad (9)$$

where E_{geom} measures the squared distance of vertices in M^m to the simplified mesh M^i, E_{scalar} measures the difference between scalar values at each vertex in M^m and the estimated values at the closest position on M^i, and E_{disc} penalizes any edge collapse in M^i that modifies a discontinuity edge with different discrete attributes on its adjacent faces.

The principal component and locking extensions are integrated directly into the progressive mesh algorithm. Scalar attribute space is reduced to p dimensions based on a viewer-specified τ. All E_{scalar} minimization occurs within this space (note that E_{geom} and E_{disc} are unaffected by this change). The selection of a new vertex position during an edge collapse $(v_i, v_j) \rightarrow v_i'$ is restricted by vertex locks; if either v_i or v_j is locked, v_i' snaps to that endpoint; if both v_i and v_j are locked, the edge collapse is not allowed.

The progressive mesh algorithm begins by computing ΔE (and an associated optimal position for the surviving v_i') for every valid edge collapse in M^m. The edge collapses are placed on a priority queue ordered by increasing ΔE. The frontmost edge collapse (with lowest ΔE) is applied, and the priorities of edges in the neighborhood of the collapse are updated. The next collapse can then be selected from the front of the queue.

4.2 Quadrics with Integrated Geometry

Garland and Heckbert [3, 4] use quadric error matrices to define a sequence of edge collapses that minimize error during simplification. A vertex v_i is characterized by the planes p_t that contain the triangles incident at v_i. The error associated with moving v_i to a new position v_i' is defined as the sum of the squared distances of v_i' from each of the planes, that is:

$$\begin{aligned} \Delta E &= \sum ({p_t}^T v_i')^2 \\ &= {v_i'}^T (\sum p_t {p_t}^T) v_i' \end{aligned} \quad (10)$$

where $p_t = [a \;\; b \;\; c \;\; d]^T$ is a vector of coefficients for the t-th plane representing v_i. The quadric error matrix for v_i is defined as $Q_i = \sum p_t {p_t}^T$. This error metric can be generalized to include scalar surface properties s_i by moving from $\mathbb{R}^3$ to $\mathbb{R}^n$; each v_i now encodes (x, y, z) and $(n-3)$ scalar values. In this domain the vertices (v_i, v_j, v_k) of a triangle form a plane $p_t \in \mathbb{R}^n$. An $(n+1) \times (n+1)$ quadric error matrix can be constructed in a manner similar to Eq. 10 to measure the squared distance of a repositioned vertex v_i' from the plane. This distance captures both geometric error and errors in estimated scalar surface properties.

During the collapse of edge (v_i, v_j) a new position v_i' must be selected. An optimal position that minimizes error can be found by solving ${Q'}^{-1}$, where Q' is the upper-left $n \times n$ submatrix of Q. Since inverting the matrix is expensive (particularly as n grows), selection can be restricted to the endpoints and their midpoint; the position that produces the smallest error is used to locate v_i'.

Using principal component space to represent scalar attributes reduces the size of the quadric error matrices to $n = p + 3$, where p is the number of principal component axes (based on a viewer-chosen τ), and the remaining three entries store the 3D geometry of the mesh. This produces a significant speed-up, particularly when computing the matrix inverses needed to find optimized vertex positions. Vertex locking forces a reposition vertex v_i' to locate at v_i or v_j (depending on which endpoint is locked); an edge with both of its endpoints locked cannot be collapsed.

The quadric error algorithm begins by computing Q_i for each v_i in the initial mesh M^m. All possible edge collapses (v_i, v_j) are identified, a new position $(v_i, v_j) \rightarrow v_i'$ is selected, and the estimated error $\Delta E = {v_i'}^T (Q_i + Q_j) v_i'$ is calculated. As in Hoppe, valid edge collapses are placed on a priority queue ordered by increasing ΔE, and removed one by one (with appropriate updates to neighboring edges) to form an optimal sequence of edge collapses.

4.3 Quadrics with Separated Geometry

Hoppe proposed modifications to the quadric error metric introduced by Garland and Heckbert to improve storage cost, execution time, and accuracy [8]. Rather than measuring the distance between v_i' and its projected position on a hyperplane in $\mathbb{R}^n$, error is computed in two steps: (1) v_i' is projected to the closest point p' in $\mathbb{R}^3$ to measure geometric error, and (2) interpolated scalar values s' at position p' are compared to v_i' to measure surface property errors. Locating p' is identical to Garland and Heckbert's original algorithm [4]. A new function:

$$\hat{s}_j^f(p) \quad = \quad {g_j^f}^T p + d_j^f, \; j = 1, \ldots, n-3 \quad (11)$$

is defined at each mesh face f for every surface property j; $\hat{s}_j^f(p)$ interpolates the scalar values stored at f's vertices, and ensures that $\hat{s}_j^f(p) = \hat{s}_j^f(p'), \; p \in \mathbb{R}^3$ (that is, any p returns the scalar value at its closest projected point p' on the underlying face). The resulting quadric error matrix:

1. Reduces storage cost, since the number of coefficients for each matrix is linear in n (previously, it was quadric in n).

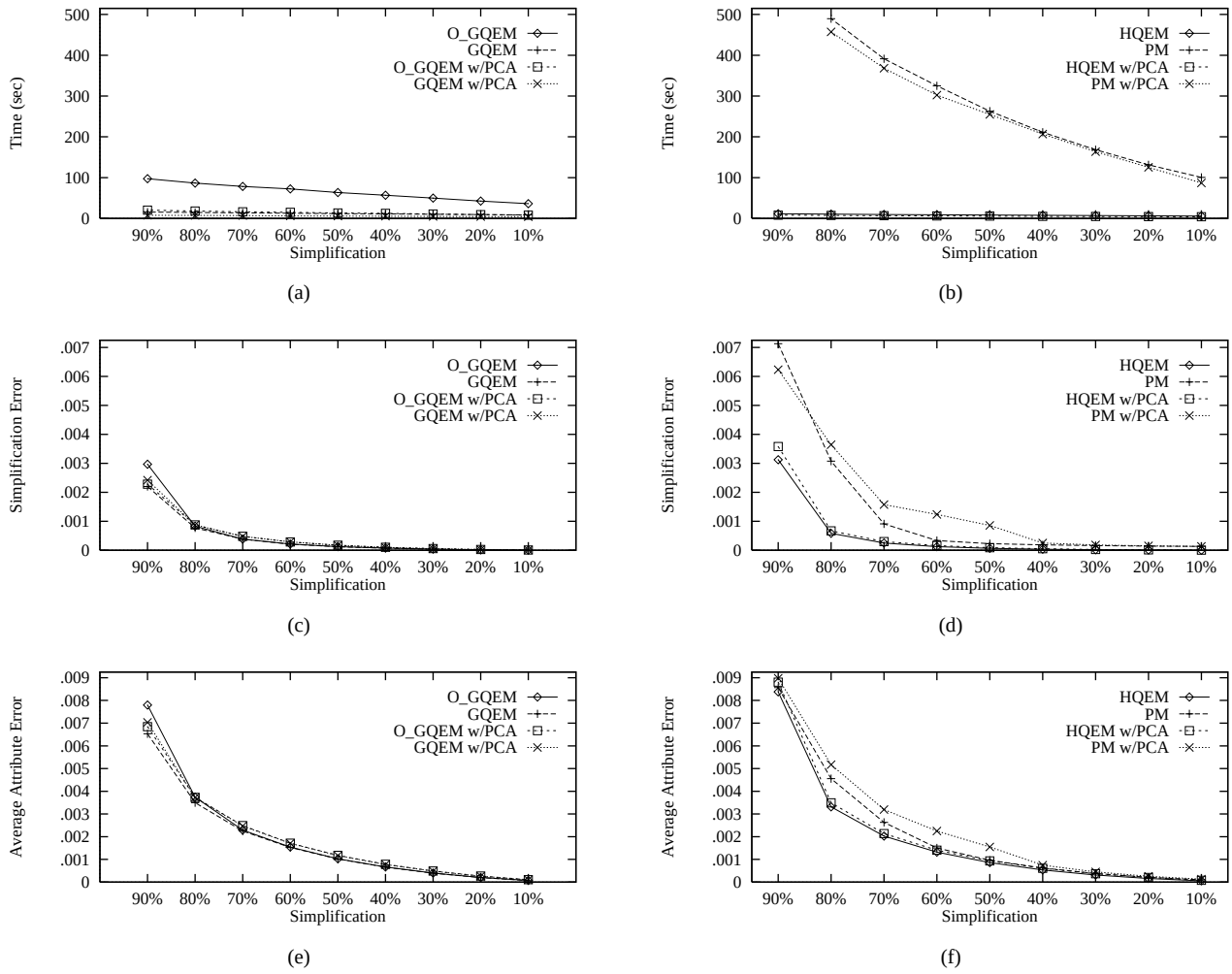


Figure 2: (a) Execution time t versus level of simplification l for optimized and non-optimized integrated quadrics (O_GQEM and GQEM, respectively), both with and without the principal component analysis (PCA) extensions; (b) t versus l for separated quadrics and progress meshes (HQEM and PM, respectively); (c, d) simplification error versus l; (e, f) average attribute error versus l

2. Reduces execution time, since the matrix is sparse.
3. Improves accuracy, since the new error metric explicitly selects the geometrically nearest position on the mesh (previously, a geometrically farther point with closer attribute values might have been selected).

Principal component analysis is integrated into Hoppe's algorithm in a manner similar to the original quadric technique. The use of p principal component axes reduces the number of $\hat{s}_j^f$ to p and the size of the quadric matrix to $n = p + 3$. As before, each edge collapses is monitored to ensure it does not attempt to reposition a locked vertex.

5 Real-World Results

In order to test our extended simplification algorithms in a real-world context, we turned to a collection of monthly environmental and weather conditions. This dataset contains mean monthly surface climate readings in $\frac{1}{2}^\circ$ latitude and longitude steps for the years 1961 to 1990 (*e.g.*, readings for January averaged over the years 1961-1990, readings for February averaged over 1961-1990, and so on). We selected eleven attributes for visualization: mean temperature *(temp)*, vapor pressure *(pressure)*, wind speed *(wind)*, wet day frequency *(wet_day)*, radiation *(radiate)*, precipitation *(precip)*, minimum temperature *(min_temp)*, maximum temperature *(max_temp)*, ground frost frequency *(frost)*, diurnal temperature range *(diurnal)*, and cloud cover *(cloud)*. Each data element e_i contains $n = 14$ values: the eleven attributes listed above, plus latitude, longitude, and elevation (a *month* was also associated with each e_i, but it was ignored during simplification).

We initially chose to visualize environmental conditions over North America. This produced a starting mesh M^m with $m = 10,056$ elements for each of the twelve months. We built separate principal component axes for each month. Although it is possible to build a single set of axes for the entire dataset, our method avoids inter-month variation, producing smaller p. For example, principal component analysis for January with $\tau = 0.9$ reduced the attribute space to $p = 4$ axes ($\lambda_1 = 0.544$, $\lambda_2 = 0.209$, $\lambda_3 = 0.105$, and $\lambda_4 = 0.068$). Results for the other months were similar (in all cases, $p = 3$ or $p = 4$ for $\tau = 0.9$). If the entire dataset is analyzed in a single computation, $p = 5$ axes are needed to satisfy $\tau = 0.9$.

Figs. 2a–d plot execution time and average simplification error (the squared distance between the original mesh M^m and the simplified mesh M^i in $\mathbb{R}^{p+3}$) as a function of the level of simplification. Figs. 2e–f plot average attribute error (the difference $(\sum_{k=1}^{n} |v'_{j,k} - v_{j,k}|)/n$ between the estimated attribute values at

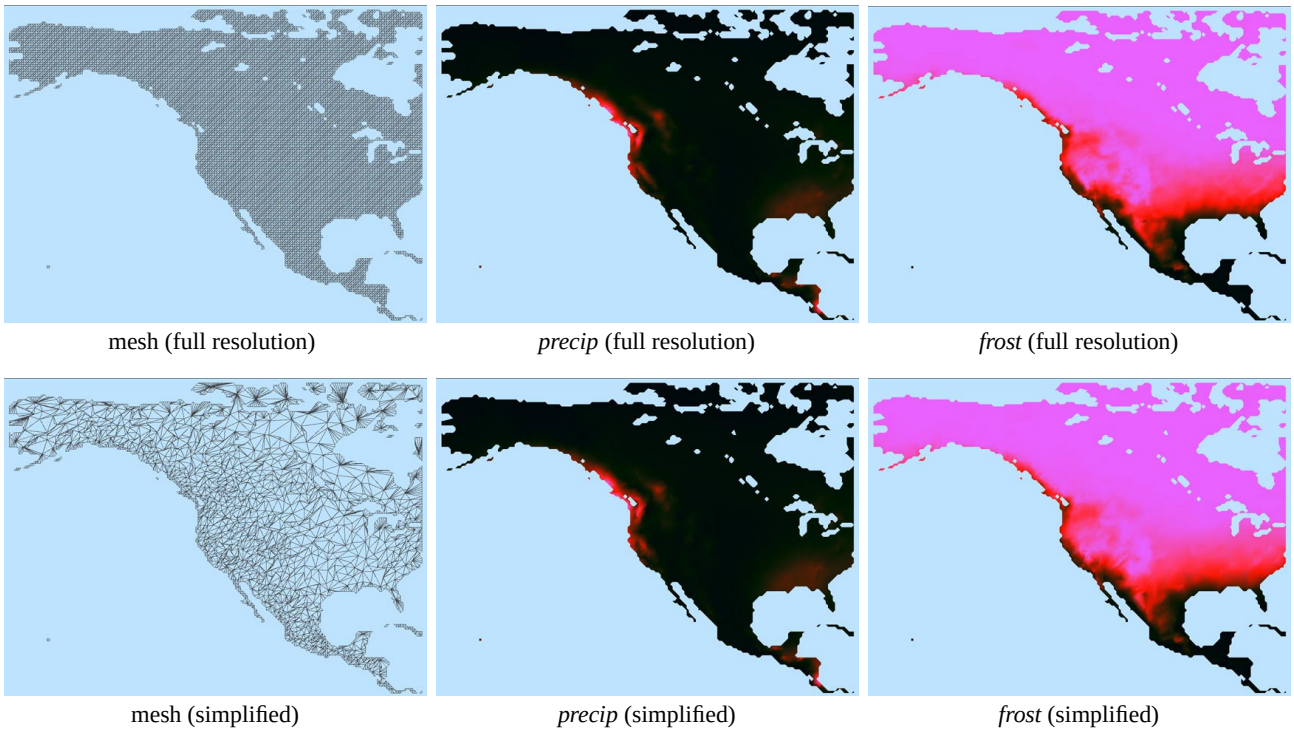


Figure 3: Visualizing the full resolution elevation height field and perceptual color scales (dark blue for small values to bright pink for large values) of *precip* and *frost* (top) versus the same data simplified 90% with progressive meshes and PCA (bottom)

$v'_j \in M^i$ and the actual values at the projected position $v_j \in M^m$). Tests were conducted both with and without principal component analysis (PCA) for each of the four algorithms: integrated quadrics with optimized vertex placement (O_GQEM), integrated quadrics with midpoint–endpoint vertex placement (GQEM), separated quadrics with midpoint–endpoint vertex placement (HQEM), and progressive meshes (PM). Our results showed:

1. PCA improved execution time for each of the four algorithms (Figs. 2a–b); the relative improvement was largest for O_GQEM, and smallest for PM.
2. Although average simplification error was slightly higher with PCA (Figs. 2c–d), in most cases the difference was not significant.
3. Average attribute error was low, even with PCA (Figs. 2e–f); in no case did it exceed 0.9%.

Figs. 2a–b plot execution time t as a function of the level of simplification l. For each algorithm t decreased when PCA was applied; the improvement was largest for O_GQEM (4.7 times faster, on average), and smallest for PM (1.06 times, on average); GQEM and HQEM showed improvements of 2.07 and 1.45 times, respectively. Although the absolute differences in t are small, they represent the simplification of a compact dataset with relatively few attributes ($m = 10,056$ elements, each with $n = 14$ attributes). For the entire dataset, the absolute improvement in t increases approximately twelve-fold. Even this level of detail is sparse; datasets with daily or hourly readings are not uncommon, and would require hundreds or thousands of simplifications. Increasing the number of attributes per element would cause an additional increase in complexity. These factors highlight the cumulative nature of any reduction in t.

Figs. 2c–d plot average simplification error (the squared distance between M^m and M^i) as a function of l. Although the use of PCA increased error, a difference of more than 1×10^{-4} occurred in only a few situations with large l. These include O_GQEM at $l = 70\%$ simplification, GQEM at 80 and 90%, HQEM at 90%, and PM at 50, 60, 70, and 80% (the PM differences appear as an obvious divergence in the error curves in Fig. 2d; other variations are smaller and more difficult to detect). Unlike execution time, increases in simplification error are not necessarily tied to dataset size and dimensionality. We have observed that accuracy remains stable as τ is held constant. A higher dimensionality n may increase the number of principal component axes p needed to capture τ of the variance, but it does not result in a significant change in accuracy (indeed, this is exactly the purpose τ was meant to serve). Similarly, increasing the size m of the dataset will increase execution time, but has little or no effect on accuracy versus an algorithm that does not use PCA.

Figs. 2e–f plot average attribute error (the difference between estimated attribute values for element $v'_j \in M^i$ and the actual values at its projected position on the original mesh M^m) as a function of l. Average error was less than 1% in all cases, moreover, the increase in error when using PCA was relatively low (between 1.05 and 1.61 times). This supports our hypothesis that improved execution times can be achieved with only a minor reduction in accuracy.

5.1 Visualization

We concluded our investigations by visualizing our simplified results on-screen. Representing high-dimensional data in an effective manner is a separate, and equally important, problem in visualization. We used the color and texture techniques of Healey and Enns [6] to display different parts of our weather dataset. In particular, we wanted to see if our simplified results captured faithfully the values stored in the original, full resolution dataset.

We began by studying how accurately the simplified attribute fields characterize their original values. Results for two fields are shown in Fig. 3. The top row of visualizes the original mesh, along with full resolution versions of the *precip* and *frost* attributes. A luminance scale is used to encode the attributes: dark represents the

smallest values, while bright represents the largest. The bottom row shows the same data for a mesh reduced to 10% of its original size using progressive meshes and principal component analysis. The estimated attribute values at the simplified element positions do an excellent job of capturing variations in the original data. Visually, it is difficult to identify differences between the full resolution and simplified fields. Some faceting can be seen along the high-to-low *precip* boundary in the Pacific Northwest, and in the *frost* patterns in the southern Rocky Mountains. The images in Fig. 3 reinforce the results found in the graphs in Figs. 2e–f: average attribute errors are small, even when simplification of the underlying dataset is high.

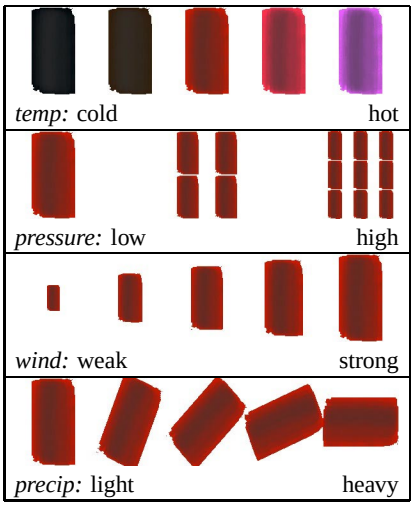


Figure 4: Examples of mapping *temp* → color, *pressure* → density, *wind* → coverage, and *precip* → orientation

We continue by visualizing *temp*, *wind*, *pressure*, and *precip* over the entire map of North America (shown both in Fig. 5 and Fig. 6). Small glyphs that look like painted brush strokes are used to represent each data element e_i. Attribute values encoded within e_i control its stroke's color and texture properties. Specifically, we mapped:

- *temp* → color: a perceptually balanced color scale represents temperature (dark blue for cold to bright pink for hot),
- *pressure* → density: the number of strokes packed into the spatial region belonging to e_i represents vapor pressure (sparse for low pressure to dense for high pressure),
- *wind* → coverage: the amount of e_i's spatial region covered by its strokes represents windspeed (low for weak winds to high for strong winds), and
- *precip* → orientation: the amount of rotation represent rainfall (vertical for light to horizontal for heavy).

Results from psychophysical experiments conducted by Healey and Enns [6] showed that proper use of these visual features will produce perceptually salient visualizations. Fig. 4 shows examples of the corresponding colors (represented by luminance for printing purposes), densities, coverages, and orientations each attribute can produce. Figs. 5 and 6 show the same mappings applied to full-resolution and simplified versions of our dataset (although we are only visualizing four environmental conditions and three geometric properties, the dataset was simplified using all $n = 14$ attributes).

Fig. 5a visualizes the original dataset (at full resolution) for January. Although this image represents certain attributes well (*e.g.*, temperature gradients are visible via luminance, and regions of low coverage with background showing through highlight areas of low wind), others are more difficult to identify. Variations in *pressure* (*i.e.*, density) and *precip* (*i.e.*, orientation) are often hard to distinguish because of strokes overlapping with one another over much of the map. A secondary concern is rendering speed: displaying $m = 10,056$ texture-mapped strokes can push refresh rates below the acceptable minimum for effective interactivity.

Fig. 5b visualizes the same data simplified to 10% of its original size using HQEM and principal component analysis. This demonstrates our ability to simplify in regions with similar geometric and environmental values, while maintaining detail in regions where one or both properties vary sharply. For example, the high concentration of tilted strokes in the Pacific Northwest capture the heavy precipitation found in this part of the continent during January (see the center column of Fig. 3 for a view of *precip* in isolation). Geographic features can also be seen as denser collections of strokes, for example, the Rocky Mountains in the west, the Appalachian Mountains in the east, and the Sierra Madre Occidental and Sierra Madre Del Sur in Mexico. The visualization also shows a higher overall density of data (*i.e.*, less simplification) in the western half of the United States. This is due not only to variations in elevation, but also to attributes with high spatial frequency components in these regions. For example, *wind* values vary significantly as they pass over the mountains; this is shown as a change in stroke size (smaller to larger) running west to east from the California coast, and north to south from northern British Columbia along the Rocky Mountain range.

Fig. 5b also highlights the problem of *planing* that can occur at high levels of simplification. When every data element is free to be relocated or removed, empty regions with very few elements can form in areas with near-constant geometry and attribute values (*e.g.*, in Nunavut and the Northwest Territories of Canada, or in the central plains and southeastern coast of the United States). Although it is useful to see where these regions occur, viewers also need a way to determine *which* attribute values were present prior to simplification.

Fig. 6a applies the same simplification techniques used in 5b, but adds vertex locking. A regular $2.5^\circ \times 2.5^\circ$ grid of data elements is locked to prohibit their relocation or removal. This underlying array is clearly visible as a sparse, regular, repeating pattern of data elements, particular in areas that previously caused planing. Fig. 6a demonstrates the final results of our work, a technique that: (1) uses feature preserving mesh simplification to significantly reduce the size of a large, multidimensional dataset in an efficient manner, (2) preserves high-variance areas of interest, (3) allows the protection of individual data elements as needed, and (4) guarantees a minimum density of data to visualize all parts of the dataset. A final example of these techniques is shown in Fig. 6b, which visualizes simplified *temp*, *pressure*, *wind*, and *precip* during January over Europe and Asia.

6 Conclusions

This paper describes a method of applying feature preserving mesh simplification to the problem of managing large, multidimensional datasets in scientific visualization. Algorithm-independent methods were developed to allow existing simplification techniques to address properties unique to our problem environment. Principal component analysis is used to temporarily reduce the dimensionality of a dataset during simplification; this ensures the algorithms continued to produce accurate, high-quality results in a time efficient manner. Vertex locking is applied to protect "important" data elements from relocation or removal, and to guarantee a minimum density of information in the simplified dataset. We used our new techniques to simplify a large, multidimensional weather dataset. Our results confirmed that: (1) the algorithms continued to gener-

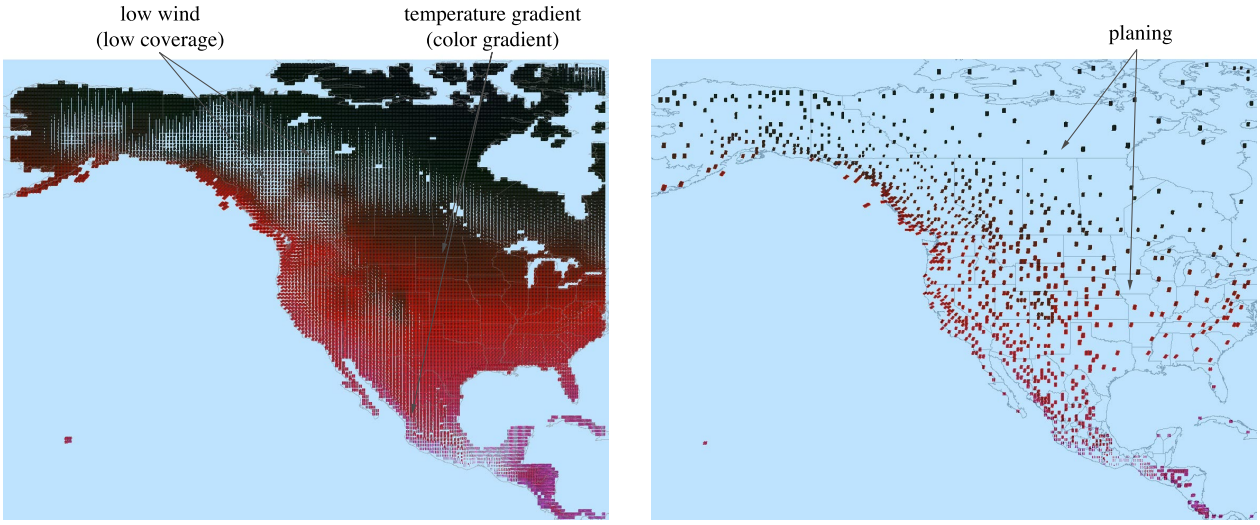


Figure 5: Visualizing *temp*, *pressure*, *wind*, and *precip* over North America: (a) full-resolution dataset; (b) dataset simplified 90% with HQEM and PCA, but no vertex locking

ate accurate results, (2) principal component analysis allowed the algorithms to execute more efficiently, particularly in the presence of large amounts of high dimensional data, and (3) any increase in error relative to the original, unmodified algorithms was small. Our simplification techniques allow us to maintain efficient runtime performance without sacrificing the high level of accuracy demanded during visualization. We are confident the same methods will be applicable to future simplification algorithms with specific properties that make them attractive to a multidimensional visualization domain.

Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation through research grants NSF-ACI-0083421 and NSF-IIS-9988507.

References

[1] BAJAJ, C. L., AND SCHIKORE, D. R. Error-bounded reduction of triangle meshes with multivariate data. In *Proceedings Visual Data Exploration and Analysis III* (1996), vol. 2656, SPIE, pp. 34–45.

[2] COHEN, J., OLANO, M., AND MANOCHA, D. Appearance-preserving simplification. In *SIGGRAPH 98 Conference Proceedings* (Orlando, Florida, 1998), M. Cohen, Ed., pp. 115–122.

[3] GARLAND, M., AND HECKBERT, P. S. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings* (Los Angeles, California, 1997), T. Whitted, Ed., pp. 209–216.

[4] GARLAND, M., AND HECKBERT, P. S. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization '98* (Research Triangle Park, North Carolina, 1998), pp. 263–269.

[5] HEALEY, C. G. One the use of perceptual cues and data mining for effective visualization of scientific datasets. In *Proceedings Graphics Interface '98* (Vancouver, Canada, 1998), pp. 177–184.

[6] HEALEY, C. G., AND ENNS, J. T. Large datasets at a glance: Combining textures and colors in scientific visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 145–167.

[7] HOPPE, H. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings* (New Orleans, Louisiana, 1996), H. Rushmeier, Ed., pp. 99–108.

[8] HOPPE, H. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings Visualization '99* (San Francisco, California, 1999), pp. 59–66.

[9] MCCORMICK, B. H., DEFANTI, T. A., AND BROWN, M. D. Visualization in scientific computing—a synopsis. *IEEE Computer Graphics & Applications 7*, 7 (1987), 61–70.

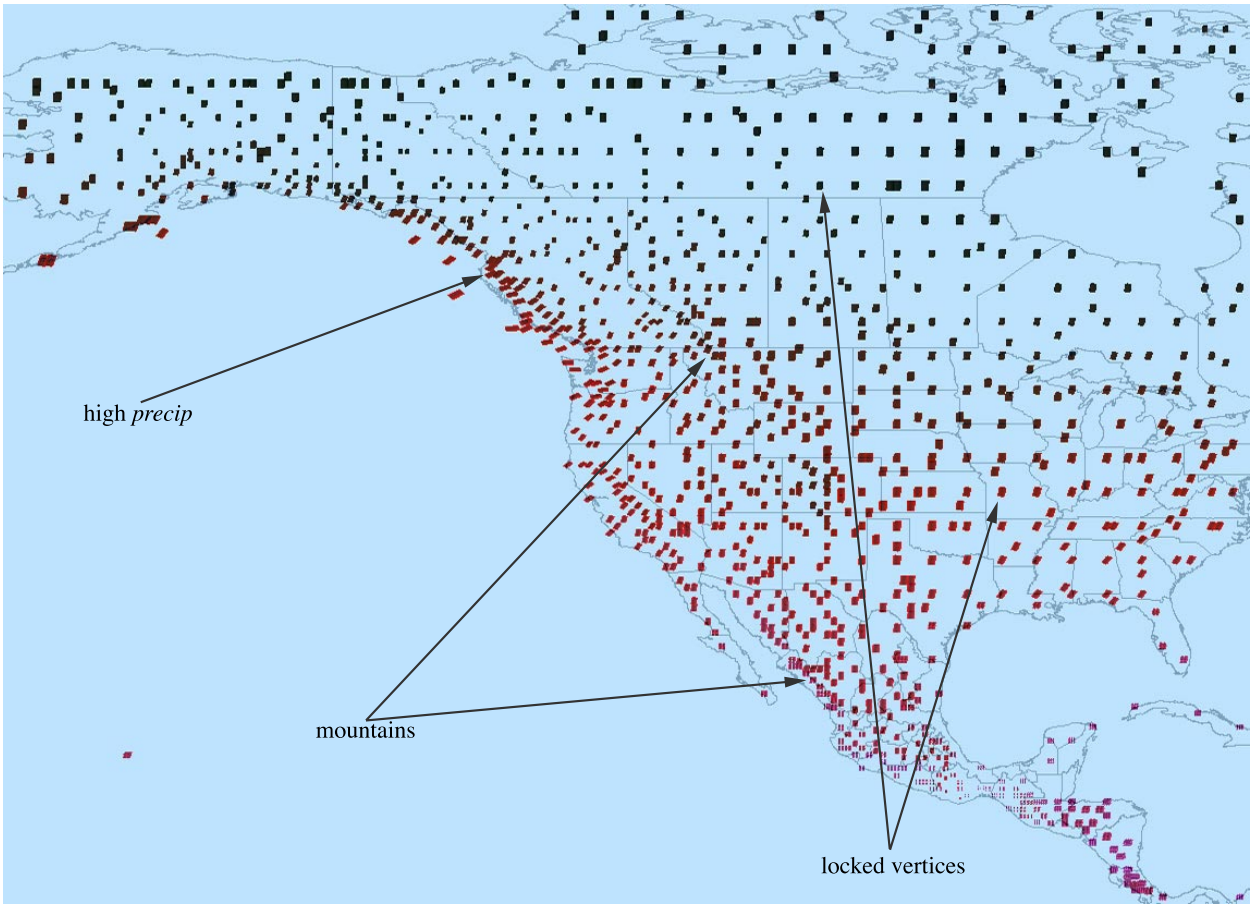
[10] ROSENBLUM, L. J. Research issues in scientific visualization. *IEEE Computer Graphics & Applications 14*, 2 (1994), 61–85.

[11] ROTH, M., AND PEIKERT, R. A higher-order method for finding vortex core lines. In *Proceedings Visualization '98* (Research Triangle Park, North Carolina, 1998), pp. 143–150.

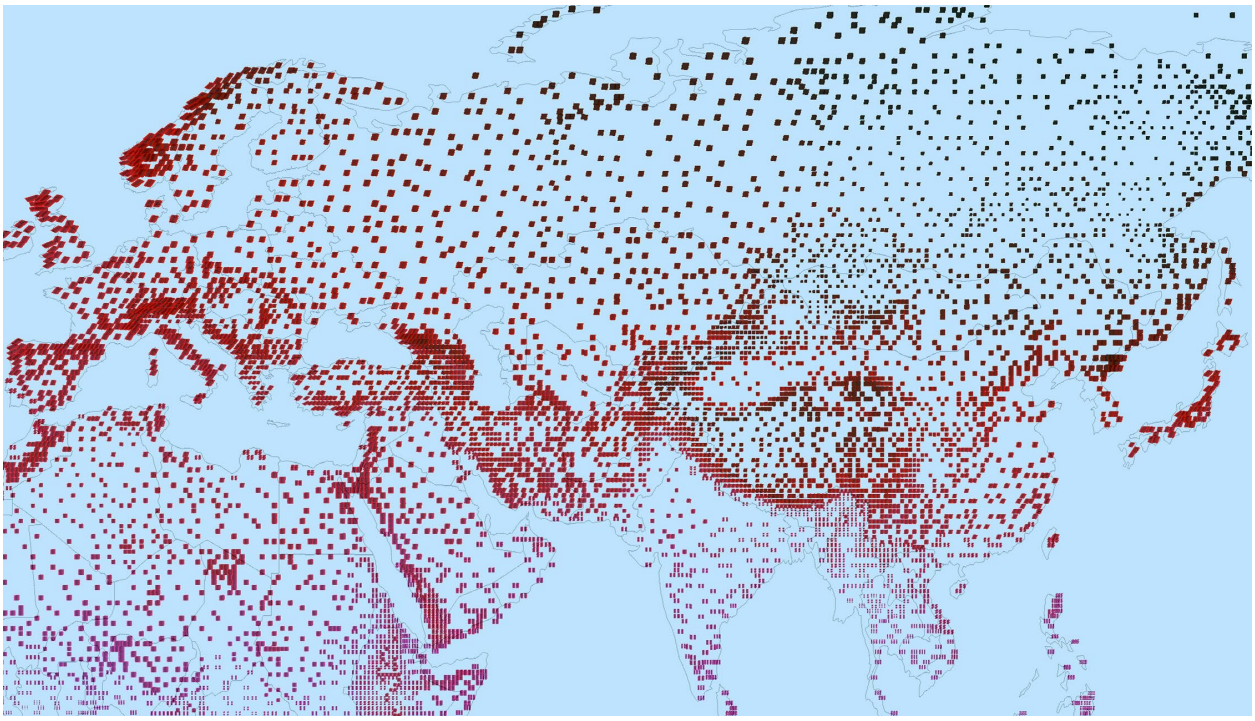
[12] SMITH, P. H., AND VAN ROSENDALE, J. Data and visualization corridors report on the 1998 CVD workshop series (sponsored by DOE and NSF). Tech. Rep. CACR-164, Center for Advanced Computing Research, California Institute of Technology, 1998.

[13] STONEBRAKER, M., CHEN, J., NATHAN, N., PAXSON, C., SU, A., AND WU, J. Tioga: A database-oriented visualization tool. In *Proceedings Visualization '93* (San Jose, California, 1993), pp. 86–93.

[14] TREINISH, L. A., AND GOETTSCHE, T. Correlative visualization techniques for multidimensional data. *IBM Journal of Research and Development 35*, 1/2 (1991), 184–204.



(a)



(b)

Figure 6: Visualizing *temp*, *pressure*, *wind*, and *precip:* (a) North America dataset with $m = 10,056$ simplified 90% with HQEM, PCA, and vertex locking; (b) Eurasia dataset with $m = 30,640$ simplified 80% with HQEM, PCA, and vertex locking