ABSTRACT

WALTER, JASON DAVID. Appearance Preserving Data Simplification. (Under the direction of Dr. Christopher G. Healey.)

Many visualization environments constantly face the issue of dealing with large, complex datasets. Often these datasets are so complex that rendering a visualization would seem impractical. Likewise, enormous amounts of data may overwhelm the human visual system; thereby rendering the data incomprehensible. Thus, the need arises to deal with these datasets in some arbitrary manner such that the resulting dataset represents the original whole — while reducing the cost on the human and computer visual system.

A closely related problem can be found in geometric models, typically represented as a piecewise linear collection of connected polygons (a mesh). Meshes can be obtained from range scanners or created with a computer aided design package. However, these obtained meshes are often very dense and have high spatial frequency. An active area of computer graphics research is directed at the simplification of these dense meshes. Initially, mesh simplification research aimed at preserving only the topology, but the most recent research, appearance preserving mesh simplification, is aimed at simplification while preserving surface properties of the mesh, such as color or texture.

Our work addresses the use of appearance preserving mesh simplification in a data simplification environment, as well as, the issues of doing so. As a result, we present and demonstrate a general method to simplify large multidimensional datasets using any appearance preserving mesh simplification algorithm. We add the use of principal components analysis to reduce the dimensionality of the data prior to simplification, which allows faster simplification on high dimensional data, and despite the reduction in dimensionality we have shown full preservation of key features in the dataset. In addition, we introduce spatial locks to preserve important data elements during the simplification process.

APPEARANCE PRESERVING DATA SIMPLIFICATION

by

Jason D. Walter

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

2001

APPROVED BY:

Christopher G. Healey

Chair of Advisory Committee

BIOGRAPHY

Jason David Walter was born on February 15, 1977 in Oil City, Pennsylvania. He received his Bachelor of Science degrees in Mathematics and Computer Science from Mount Union College, Alliance, Ohio. After completing his undergraduate work in 1999, he entered graduate school at North Carolina State University in pursuit of a Master of Science degree in Computer Science. On March 27, 2001, Jason successfully completed his oral preliminaries and plans to graduate in May. After graduation, he expects to continue his career within industry while actively furthering his education.

ACKNOWLEDGEMENTS

There are many people that I need to thank. This thesis would not be possible without them. First, I would like to thank my committee members, Dr. Christopher Healey, Dr. Robert St. Amant, and Dr. James Lester, for their time and attention during completion of this thesis. A special thanks goes to my advisor, Dr. Healey, for his ever so welcome advice and candor.

I would also like to thank a number of friends and peers for their "sanity checks" during the writing of this document: Sarat, Brent, Laura, Jiae, Mike, Mark, Ryan, Mahmoud, Louisa, and Tyler. Also, a special thanks goes to my girlfriend, Samantha, who gave me unconditional support and patience throughout my graduate work.

Finally, I especially want to thank my parents and family whose direction and guidance have helped me achieve what I am today. And, to all those people I meet, life is only possible together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Visualization is a way of understanding arbitrary data. In its simplest form, we are interested in understanding a single entity with respect to another. More complicated visualizations make use of low level properties of human vision to build preattentive visualizations that allow users to rapidly "realize" key features in a dataset. These preattentive visualizations not only allow a particular user the ability to rapidly and effectively visualize data, but it also allows the user to analyze multidimensional data.

Unfortunately, as our tools used to examine data improves so does the complexity and size of a given dataset. And, despite our best efforts, large multidimensional datasets are becoming increasing more prevalent in all aspects of scientific visualization. Some examples include, geographic information systems, land and satellite weather information, and scientific simulations. With the advent of machines with faster hardware and more memory it would seem unreasonable not to use a visualization system to analyze a set of data. But there becomes a point when, regardless of the type of system you are using, data becomes too large or complex for a system to process in a reasonable amount of time. In many situations, these datasets are so large and complex that rendering a visualization on even the most high-end systems is impossible, and, even if a visualization can be rendered, it often overwhelms the user with so much visual information that it is nearly incomprehensible.

Many have acknowledged the problem of large datasets. For instance, Silbershatz et al. [SSU90] make specific reference to the problem of large scientific datasets, and the need to find new ways to store and query them. Also, a set of end-users, in physics, mention the proliferation of large, multidimensional datasets and the inability to manage, explore, analyze, display, or discover within them [BGK+88]. Moreover, McCormick et al. [MDB87] expressed the need for dataset management due to its enormous size, and, recently, van Dam et al. [vDFL+00] discussed the most pressing issues of scientific visualization and reminds us that new techniques are needed to deal with large multidimensional datasets.

Typically, the "size" of a dataset refers to one or more of the following qualities [Hea98]:

- *Number*: the number of data elements participating in the dataset.
- *Dimension*: the attributes associated with each data element, e.g. temperature, pressure, wind speed, etc., at a spatial location.
- *Range*: the range of values a particular attribute can contain, e.g. $[0, 1]$ using single precision, four byte, floats.

Each of these properties may need to be considered when visualizing a dataset. Naturally, a slight change in one quality can drastically affect the size of a dataset. For example, consider a terabyte dataset. If we added an additional attribute per data element we have just doubled the entire size of the dataset. Therefore, it is essential to derive new ways to deal with these large datasets.

Most of the recent work in the handling of large data can be categorized in two ways:

- *Data management*: how can we store large amounts data, and how can we effectively and efficiently retrieve "important" information?
- *Data simplification*: can we find areas in a dataset that have little significance to the total dataset and remove them?

Researchers who are interested in data management techniques in scientific visualization work closely with researchers in the field of databases. These systems typically make a bridge between a particular database system and a visualization engine. Here users can query specific parts of a massive dataset, with reasonable access time. The work done by Michael Cox and David Ellsworth [CE97] is a classic example of one these systems. They overview ways to bring the computation to the data, and they take advantage of memory hierarchy to handle large data. Also, they emphasize that given big data we must sacrifice components of visualization (i.e. interactivity).

Others who are interested in data simplification tend to focus on ways to classify the data as unnecessary and have it removed. These methods cull large amounts of data with high spatial frequency to allow a greater quantity of the data to be visualized. These types of techniques are useful; as they curb the load on the visualization system, both graphically and computationally, and, in turn, reduces the visual load on the user.

1.1 Main Contribution

As we agree with Cox and Ellsworth [CE97] that the most important goal to visualization is to understand the data. We also believe that a main culprit in large data is redundancy that can be eliminated by some type of simplification. In fact, the Department of Energy and the National Science Foundation have recently expressed interest in financing efforts in data simplification using AI techniques. They want "AI techniques that cull the original dataset to ∼1% of the most interesting entries"[SR98].

We approach the problem of data simplification in a slightly different perspective. First, consider a similar problem that occurs commonly in computer graphics applications, i.e. geometric models that have a high spatial frequency are difficult to render, if not impossible. In lieu of this, there is a substantial amount of work that is devoted to dealing with large data models. Specifically, this work addresses simplifying these models such that the simplified models still retain a majority of the necessary information, i.e. topology and attribute surface properties. Now note that in scientific visualization we have a similar problem, i.e. large amount of data with high spatial frequency, where attribute information is assigned at a spatial location. Therefore, it would only seem appropriate to use mesh simplification to simplify data. However, the solution is not as simple as it seems. Many of these feature preserving simplification algorithms assume that the data to be simplified is a geometric model, thereby limiting scope to only a few attributes, e.g. color and normals. Because of this assumption, consideration is not given to high-dimensionality, where the number of attributes may be large. For example, a dataset may have as many attributes per data element as simple geometric object has vertices. It is not that a particular algorithm is not capable of working with more than just color or texture; it is simply that it may not be feasible to do so. Additionally, many algorithms employ some type of vertex relocation to create a "best fit" to the original model. However, vertex relocation may not be desired in a visualization environment. Regardless, mesh simplification offers a plausible solution our problem of simplifying large data. In this paper, we examine the use of mesh simplification in a data simplification environment with the following goals:

- *General*: Mesh simplification has been an active area of research, and thus it is necessary that a visualization system not be constrained by a particular method.
- *Computationally Sound*: In addition to the computational complexity of a particular method, any additional overhead needed for that method should be insignificant with respect to the method. And if a particular method's computational complexity explodes when introduced to large multidimensional datasets, then it is necessary to reduce the computational load.
- *Space Efficient*: Because we are working with very large datasets, it is necessary to reduce, or limit, the space overhead (in-core), in addition to a particular method's overhead, without imposing on our second criteria.

Our system is an implementation of three recent algorithms in mesh simplification. We apply these algorithms to arbitrary data and employ a dimension reduction technique, principal components analysis, to improve the performance and feasibility of the algorithm in a data simplification environment. To illustrate the effectiveness, we compared results, visually and quantitatively, before and after simplification, with and without principal components analysis. We discovered that by using principal components analysis we can considerable reduce the size of a dataset and improve system simplification performance. We also found that, despite a loss in dimensionality, key dataset features still remained and little information was lost from the change in dimension. Furthermore, principal components analysis offered a general extension to any type of simplification algorithm, and, most importantly, we satisfied the three conditions that we initially imposed. Each of these cases will be illustrated with numerical and visual results, from the software that we built.

In addition, we applied locks to the dataset to emphasize important features of the dataset. This especially useful in a point based visualization system, where the underlying mesh may not be important to the domain.

1.2 Paper Organization

The remainder of the paper will proceed as follows. In Chapter 2, we review background and related work in the area of scientific visualization. Chapter 3 will review the most recent work in mesh simplification. Chapter 4 will address feature-preserving data simplification, with our designed extensions. Chapter 5 will focus on implementation of our method in specific algorithms and results. Finally, future work and concluding remarks are mentioned in Chapter 6.

Chapter 2

Arbitrary Data and Scientific Visualization

A little over ten years ago, Richard Saul Wurman wrote that the amount of "human information doubles every 5 years." Even today we are aware of this information explosion, perhaps moving much faster than the Wurman's original conjecture. And many suggest that by the year 2020, the amount of human information that surrounds us will double every 73 days [Tra01]. Current estimates place the total amount of worldly information at about 1.5 billion gigabytes of data, and, with this exponential rate of growth, the amount of data in the 2020 will be far beyond anyone's comprehension. Undisputedly, at the forefront of this growth lies the computer.

Scientists in a variety of different fields depend on computers to better their understanding of many difficult problems in science. Typically a scientist will not know the significance or relationship between a large set of numbers. In fact, they often seek to find a correlation that will help them better understand what they are looking at. The advent of graphical displays allows scientists to map acquired numerical information to a visual feature that establishes significance to the dataset; thereby giving birth to scientific visualization. For example, let us consider a typical entomologist. Suppose this scientist, who we will call Henry, wants to see if their is a relationship between elevation of a location versus the mosquito population at the location. Henry samples mosquito population at discrete locations and plots them on his computer, assigning color to the varying levels of mosquito population. Immediately, Henry discovers that there is some relationship, but realizes he still has work ahead of him.

In this over-simplified example, we can see the simplest form of visualization, and we can see the breadth at which a visualization can apply. Scientific visualization is not just something for computer scientists to work on; it is "embedded in the scientific and engineering research process" [HPvW94]. It is a tool that combines work from many disciplines, and it has implications in many more. Meteorologist, physicists, sociologists, biologists, chemists, and many others depend on visualizations to advance progress in their fields. And scientific visualization is necessary as our wealth of information grows far beyond what has ever be comprehended; as our society develops into an advanced civilization.

Surprisingly, the notion of scientific visualization is relatively new, only first mentioned by McCormick in 1987 [MDB87], and the study of visualization in computer science is just as old. But the notion of visually representing numerical data has been around for some time, e.g. cartographers and artists are masters in the creation of visually rich and effective illustration [Tuf93].[1] It usually took these artisans considerable time to generate an acceptable visualization, but they were well aware of how the human visual system perceived images. Therefore they were capable of generating very effective visualizations.

In essence these artisans have some notion of mapping. They can map numerical or conceptual data into a physical form. Or stated mathematically,

$$g : presentation \Leftarrow data \tag{2.0.1}$$

where g is a function that tells the artist what to do when he sees $x \in data$. For example, an artist working on a visualization of ocean currents may draw lines in the direction of the current and use the density of his pen stroke to represent the current strength of that flow [Tur96]. Another artist may choose to use a variance of color to represent height on a topological map, creating what is known as a height map. Regardless, the notion remains the same: take a data value and map that value to a visual feature [Hea92]. Today, these mapping can be performed with much more speed and ease than traditional methods, i.e. with scientific visualization software packages. These packages make use of what is known about salient visual mappings that can create effective visualization; providing users with their own personal artist — minus the creativity. The visualization engine simply applies a similar mapping as in (2.0.1).

Now g can vary across the spectrum of scientific visualization software packages. At best, most of the previous attempts in visualization were very ad hoc, rules-of-thumb based,

[1] We will refer to cartographers and artists as just simply artists.

and were catered to specific applications [Ros94]. Recent attempts have been more general [Hea92, WEL+00, CL93, vW91]; focusing on finding a g that is applicable across a spectrum of scientific visualization needs. Specifically there are a broad range of datasets that are fundamentally generic, and, therefore, we only need to find a suitably generic g. These techniques are our primary interest, and we will devote the remainder of the chapter to these "generic" methods. Many of these methods work well across the board of application environments, but their shortfall is dealing with extremely large datasets. It is not that they are not acceptable methods; it is just that were not designed with the concept of very large datasets in mind.

On this note, it is interesting to review the tone of research issues in the field of scientific visualization over the past 14 years. This review will emphasize the development of scientific visualization. It will also lay the groundwork for the motivation of our work, and it will explain why many current "generic" approaches are not designed for large data.

Most of what is scientific visualization is due, in part, to three key panels sponsored by the Department of Energy and the Department of Defense. During the first panel discussion in 1987 [MDB87], focus was geared primarily towards tool creation. Specifically, a push was emphasized on the creation of tools for end users that increased scientific progress, collaboration, and productivity. McCormick enumerates the following list of issues for tool makers:[2]

- Interactive steering of computations and simulations.
- Workstation-driven use of supercomputers.
- Graphics-oriented programming environments.
- Visualization of higher-dimension scalar, vector, and tensor fields.
- Dynamic visualization of fields and flow.
- High bandwidth networks for pictures.
- Handling terabyte data sets - for signal and image processing.
- Vectorized and parallized algorithms.

[2] This list follows verbatim from the list provided by Rae Earnshaw [BCE+92]

- Specialized architectures for graphics and image processing.
- Establishing a framework for international standard in scientific visualization.

Interestingly, little emphasis was placed on the management of large datasets. At a later panel discussion in 1994 [Ros94]. A revised list of research "issues" was announced. Among the most pressing issues at this workshop included but are not limited to:

- Dealing with large datasets in volume visualization.
- Interactive scientific visualization on public networks.
- Compression algorithms for transfer of scientific visualization.
- Scientific visualization in portable computing environment.
- Improving data modeling techniques, e.g. better segmentation techniques.[3]
- Modeling of complex objects.

Here we can see more attention to the issue of large datasets, but still a large focus on other interests in scientific visualization. Of course, this is easily understandable given the infancy of scientific visualization. However, this tone drastically changed in 1998 when a third panel met to review the most pressing issues in scientific visualization [SR98]. The focus of this panel more strongly emphasized the problem with large datasets, and they iterate that "[c]urrent graphics and visualization technology cannot cope with the volume or complexity of the data produced by simulations that will be carried out ... in the next two to four years" [SR98]. Moreover,

- Visualizations are nearly a necessity when understanding the "results of large-scale scientific simulations."
- New data management techniques are necessary.
- New systems are necessary that provide an "end-to-end" solutions for user, including interactive manipulation of large datasets.
- Research and development activities in data manipulation and visualization techniques need to increase significantly.

[3]Segmentation is the process of finding regions that satisfy a particular feature.

This panel strongly emphasizes that large data is a problem that hinders our advancement in scientific knowledge, both present and future. And to continue our onward march, we must focus our efforts on systems that efficiently and effectively deal with large datasets. Furthermore, prior to this panel, less emphasis was placed on dealing with large datasets, and, consequently, less work was performed on data management vis-à-vis visualization tools and techniques. Of course, it goes without saying that there has been a body of work on the problem of data management.

2.1 Generic Visualization Techniques

Before we begin our discourse on current approaches to dealing with large data. We will provide some background on current, commercial-based, software packages available to researchers and scientists. However, we will only touch on a few of the most common, generic, visualization systems, illustrating the benefits of each system. We will also illustrate how these systems deal with large data, and we will show how these systems fail to "fit the bill" when it comes to dealing with the large datasets of tomorrow. Following sections will overview perceptually based multivariate visualization techniques, which is what our system is based on. Then we will present some current work in data management.

Obviously in the creation of a visualization system, a few key properties are important. Namely, you want the system to be useful, meaningful, intuitive, interactive, and responsive [Hea92]. Because we seek a general computing environment, solutions have to be general with respect to the type of problems that could be encountered. Specifically, systems provide a layer of glue that abstracts visual concepts with the corresponding data. For instance, visual concepts like spatial location, color, and structure, i.e. model structure, are typically used to map dataset features/properties to create a visual representation. Moreover, a software package must allow flexibility to create arbitrary visualizations simply and effectively, e.g. through the use of a command language. Furthermore, a software package must be responsive to user input such that he can "see" progress he/she is making as he/she produces a visualization [BCE+92].

Now there have been many commercial packages that attempt to do above. An example of such a system is AVS. Application Visualization System (AVS), later renamed to Advanced Visualization System, is a module based visualization system, where individual modules control the functionality of the system [GM92]. This system is well developed and currently in its fifth revision. The basic system consists of four main modules:

- *Rendering module*: responsible for displaying data.
- *Mapping module*: creates a mapping between data and features.
- *Processing and filtering module*: for transposing and cropping data.
- *Input module*: obtains data from an input source (via data management facilities).

For the most part, AVS provides a flexible, extensible, and meaningful environment. Specifically, AVS provides extensibility in a two ways. First, users can customize the parameters of the modules to specify how to read, filter, and display data. Second, users can create new modules through an API to add additionally functionality to the system. However, this "feature" is limited to those who know how to write code and use the API, and, therefore, lacks an intuitive feel. Moreover, AVS lacks advanced filtering tools, where producing a visualization may become hindered in the presence next generation large datasets [SR98].

Open Data Explorer (OpenDX) is an open source data visualization system that branched off of IBM's Data Explorer [IBM97]. It is hoped that by releasing this product to the open source community, OpenDX will advance to suit the needs of next generation visualization. OpenDX is client/server based, designed to run on shared multiprocessor systems. The system is modular in design and provides a drag and drop interface for visualization creation. The system also supports an advanced data model, where data of any type can be abstracted. However, this system lacks any advanced data management/simplification facilities. The set of available tools includes:[4]

- *Reduce*: Reduces the resolution of a dataset, filtering and resampling the set at a lower resolution.
- *Refine*: Increases the number of samples in a data set. This module interpolates data values or colors at the new positions from the data or color values at the original positions. It can also convert connections from quads or faces to triangles, and from cubes to tetrahedra.
- *Slab*: Takes a positional subset of a data set with regular connections (cubes, quads, and lines). You can specify along which axis to take the subset, where the slab should begin, and how thick the slab is to be.

[4] These are presented in verbatim from IBM's reference manual [IBM97].

- *Slice*: Takes a positional subset of a data set with regular connections, like the Slab module. However, unlike the Slab module, it creates output with a dimensionality one less than the dimensionality of the input. For example, you can import a data set as a 4-dimensional grid, with the fourth dimension representing time. You can then use the Slice module to create 3-dimensional slices of the data set.

None of these functions provide any advanced data culling, e.g. data mining/knowledge discovery. Moreover, functions like *Reduce* and *Slab* are useful but too restrictive (limiting functionality).

VIS-5D is another open source project, directed by University of Wisconsin Space Science Center, designed for visualizing weather models and similar data sources, i.e. explaining physical phenomenon like earthquakes. It works in a five dimensional data model, where the first three dimensions corresponds to spatial location, the fourth is the temporal dimension, and the data value of interest [HS90].

VIS-5D excels in its intended application environment. Specifically it excels at visualization data that has some earthly meaning, e.g. cloud cover, precipitation, ground water saturation levels, etc. It was designed to deal with large datasets that conform to the five-dimensional format, and it provides useful and meaningful illustrations using this model. But, unfortunately, this restriction limits the flexibility of the software and like the other packages we discussed, doesn't provide sophisticated data management and/or culling facilities.

All of these software packages are capable of producing effective visualizations in their designed environments; however, more needs work needs to be done to allow access to larger, more complex, datasets [SR98].

2.2 Multivariate Visualization Techniques

Before we overview some of the current approaches that deal with large datasets, we will provide a overview on multidimensional visualization systems. Specifically, perceptually based visualization systems like that presented by [Hea92]. We choose to overview this system here because later we will present our results using a similar system.

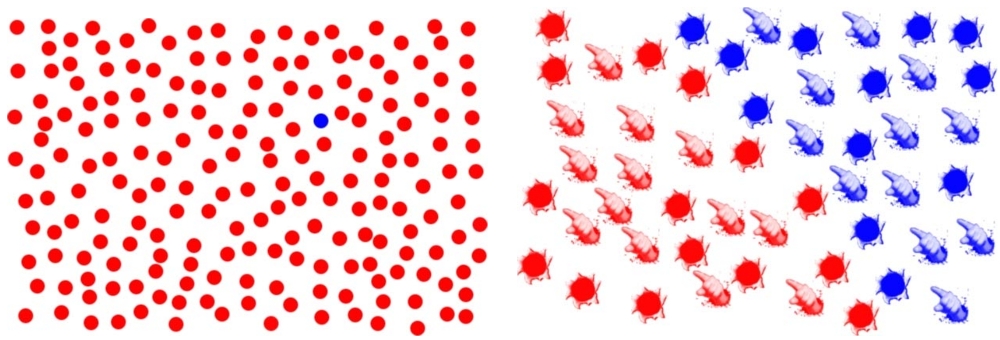
Often, it is not enough to have just single variate visualizations. Single variate visualizations, such as a temperature map or a precipitation map, are sufficient for evening news weather broadcasts, but, more often, researchers are interested in the relationship between variables. Therefore, it would be extremely useful to somehow display two, or more, quantities simultaneously, e.g. side-by-side. Specifically, we have a n dimensional data element that we wish to map to n visual features. The data element may contain weather data, such that the first three dimensions specific spatial location, the remaining dimensions may contain information regarding temperature, humidity, pressure, wind speed, etc. Regardless, we want to "visualize" all n dimensions simultaneously. More importantly, we want all of the n features to convey meaning such that no one feature masks, or occludes, another feature. These cases prevent the user from "seeing" all the data.

Naturally, this is not an easy problem, and many researchers are addressing this issue. Specifically, Healey and others [WEL$^+$00, Hea96, Hea92, Hea99] are applying concepts from cognitive psychology to create perceptually based multidimensional visualizations. These systems take advantage of the preattentive processing power of the human visual system. Preattentive tasks are those that can be rapidly, less than 200ms, processed by the brain, i.e. before visual attention. These preattentive processes handle fundamental visual tasks like edge detection, color perception, and other spatially independent task. Some of these tasks are illustrated in Figure 2.1. For instance, glancing quickly at Figure 2.1(a) on can easily notice the blue dot in the field of red dots. Also looking at Figure 2.1(b) one can quickly notice a boundary between the blue objects and the red objects. Additionally, Figure 2.1(b) illustrates the potential to combine different preattentive features, i.e. color and texture, to produce a multidimensional visualization. Therefore, if we can create a system that makes use of these visual properties — and others that have been discovered via cognitive experiments (cf. [Hea99]) — we can create an effective visualization system that allows rapid and effortless visualizations.

However, preattentive features cannot be combined haphazardly. Some preattentive features may occlude other features, and, thus, "block" a user's ability to "see" data properties. Healey has addressed this issue and has created perceptual textures.

Perceptual textures are based on the notion that texture, e.g. texture of a wall or floor, is composed of a series of texture elements and by tuning each of those elements you can, in effect, alter the appearance of the texture. Some of these properties include:

- *Luminance*: Brightness of the texture (or features within the texture).
- *Color*: Color variation in the texture.



(a) (b)

Figure 2.1: Examples of preattentive tasks.

- *Orientation*: The direction of the texture, e.g. horizontal versus vertical.
- *Size*: The size properties within a texture, e.g. large red dots viz-á-viz small red dots.
- *Density*: The spatial area between objects (i.e. large spatial gap $\rightarrow$ less dense).

As it turns out, the human visual system can identify differences in texture (cf. [Hea99]). So, Healey creates a perceptual texture element, *pexel*, that is based on the above texture properties; we can assign attribute values of a data element (in a unique spatial location) to these textural properties. Those data elements that are spatially local to one another will create a texture pattern that describes data in that region. Figure 2.2 illustrates pexels and the effects of varying the properties of that pexel (as the attribute values vary between some range, e.g. $[0, 1]$, the pexel feature changes, i.e. left to right). The underlying dataset is a weather dataset captured over the Gulf of Mexico, where color is mapped to temperature, orientation is mapped to precipitation, pressure is mapped to density, and size is mapped to wind speed. The combination of pexel properties create texture regions that allow the user to rapidly find interesting and key features. Thereby creating effective, powerful, general and useful visualizations. We make use of this visualization model in a later chapter to demonstrate our simplification extensions.

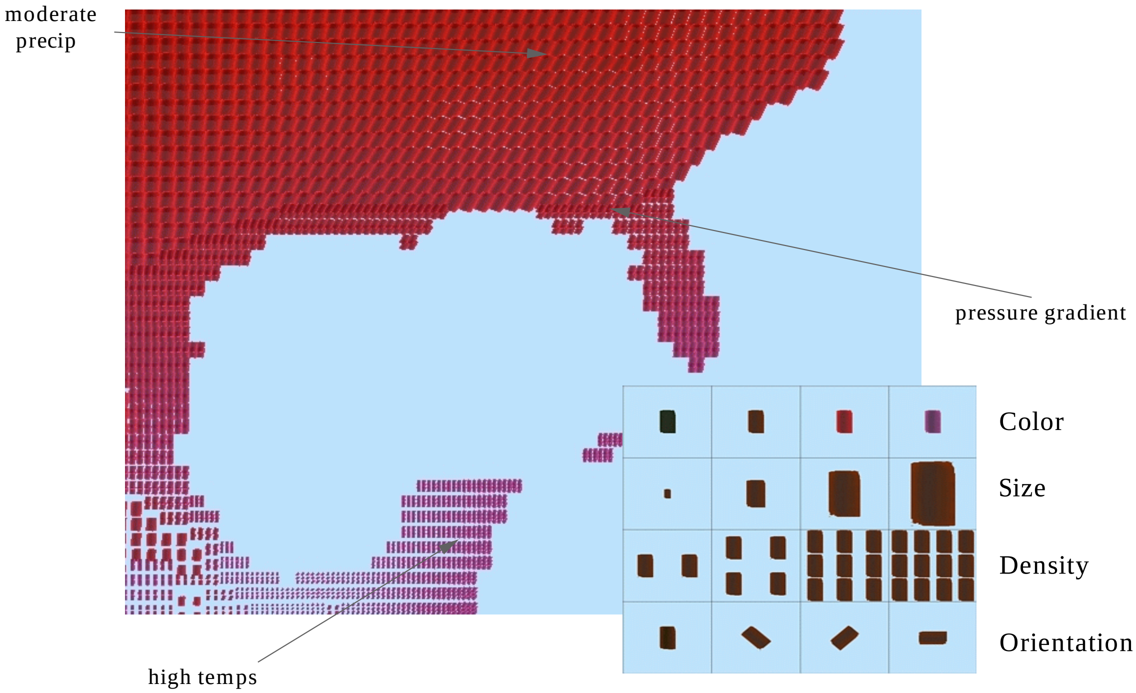


Figure 2.2: Using pexels to visualize datasets.

2.3 Current Large Data Approaches

As the amount of scientific data grows beyond comprehensible proportions, new techniques are needed to foster the feasibility of a visualization. Numerous techniques currently exist for dealing with large data, and we will review those techniques here.

As we mentioned previously, current approaches to large data visualization fall under two major categories.

- *Data management*: how can we store large amounts data and efficiently retrieve important or desired information?
- *Data simplification*: can we find areas in a dataset that have little significance to the total dataset and remove them?

The work in data management borrows heavily from research in databases These database management systems (DBMS) are attractive because they offer flexibility in data representation, and they allow access to large amounts of data seamlessly. Moreover, DBMS offer a better management strategy over previous attempts in data management, e.g. in systems like VIS-5D, AVS, and Data explorer [Hea96]. However, traditional database systems impose a number of problems [BCE+92, Hea96]:

- They are relational — databases depend on relational properties of data. Consequently, this model does not accommodate multidimensional data or the occurrence/presence of errors in a dataset. Moreover, it does not suit the spatial qualities of data.

- Unwanted Facilities — databases provide superfluous functions that are unwanted and unnecessary in a visualization system, e.g. report generation.

- Language Support — often databases do not allow extensions via scientific programming languages.

More recent developments in DBMS, like Postgres and Oracle, overcome some of these problems — in most cases providing extra glue to allow greater flexibility. As a result, attempts have been made that link these more advanced database systems with visualization systems. Tioga and Tioga-2 are examples of such systems [SCN+93, ACSW96]. These systems tightly couple a database with an arbitrary visualization system. Therein providing advanced data access/management to a visualization system; however, we are still working with a relational model.

Nonetheless, in the case of Tioga, a bridge is made to a Postgres DBMS with all functionality of Postgres available through their interface, and a language is used to bridge the visualization application to the database server. The user can build a visualization system using an interface based on traditional *boxes* and *arrows* — where the boxes represent generic data structures that allow operations on the data, and the arrows depict the flow of data from one box to another box. Within a box, the user describes the operations to perform on the data as it flows through the box, e.g. data conversion. Furthermore, boxes are not limited to strictly data operations; they can describe how the data is physically displayed on the screen.[5]

Despite the use of advanced DBMS in systems like Tioga and Tioga-2, fundamental problems still exist in these systems. Specifically, "data with noise, non-deterministic values, missing values, or errors are difficult to store" or represent "in a relational DBMS" [Hea96].

[5] Aside, visualizations built using Tioga are called *recipes*, and the boxes are the *ingredients*. User-defined Postgres functions are available to the user as ingredients

Moreover, spatial and temporal properties of a dataset does not lend itself to a relational model. And even with the generic properties of a DBMS, which allows new data types and operators, new techniques that allow spatial and temporal relationship are difficult to implement.

Other techniques have been developed that work around some of the inherit disadvantages of a DBMS. Knowledge discovery, or data mining, is one of these fields. Specifically, data mining allows the extrapolation of "interesting" data from a dataset, where areas are classified as "interesting" based on a set of user-defined rules. These rules aid a data mining system in the collection "of implicit, previously unknown, and potentially useful information from the data" [FPSM91]. These systems combine research from statistics, artificial intelligence, and database systems. In a sense, data mining combines these three fields to allow the exploration of large datasets, which are so large that they could never be explored by a human observer.

Many companies and research institutions are using data mining for a multitude of purposes. For example, some US retail chains use data mining to extract subtle shopping trends in buyers, e.g. more people buy X at Y than at Z. IBM has created a data mining tool for the NBA that allows coaches and officials to look for interesting game statistics. Bioinformatics and pharmaceutical companies are researching data mining techniques to explore the human genome for "gene-related" diseases, and a number of other companies are using data mining to perform advanced fraud detection [PS01]

Visualization systems have recently taken advantage of data mining research. For example, Koperski [KHS98] and his group have designed a spatial data mining system that combines decision trees and spatial data structures to explore geographic information systems, remote sensing systems and computer cartography (to name a few). Ester [EKX95] has done similar work in spatial classification using R*-trees to perform efficient queries. He later extends his work to handle datasets with missing and erroneous data [EKSX96]. Healey [Hea98, Hea96] successfully couples several data mining approaches with his perceptual based visualization system — his approach is a general attempt and not specific to a particular application. Regardless, data mining is used to cull unimportant data from the dataset.

Koperski interests are in the classification of data using spatial queries. His system, GeoMine, is the result of this work. Furthermore, he classified spatial queries into the following categories:

- Non-spatial attributes of objects, e.g. number of stores that sell widgets.
- Spatially related attributes with no spatial values, e.g. the number of people living 1km from the store that sells widgets.
- Spatial predicates, e.g. distance_less_than_10km(X, store).
- Spatial functions, e.g. driving_distance(X, store).

His system support queries of those types of categories, and he successfully shows that this minimum set is sufficient to perform many complex queries efficiently. Moreover, after the query is performed the user is shown visual cues illustrating the results of the query.

Healey couples his visualization system with four different, classification based, data mining techniques — techniques similar to Ester's and Koperski's work. They believe that classification algorithms can aid the visualization of large multidimensional datasets, through the discovery of unknown information. They also believe that data mining algorithms can reduce the size of large datasets. Their implementation successfully demonstrates these two benefits. They also demonstrate that data mining performs better in the presence of errors than conventional methods.

There have been many other attempts to deal with large data. On the data management side, common data formats have been constructed that allow storage of spatial based data. For example, Hierarchical Data Format (HDF) was developed by the National Center for Supercomputer applications, University of Illinois, at Urbana-Champaign to enable application independent data storage/exchange. Because HDF is a spatially organized format, an application can pull specific regions out of the dataset. Common Data Format (CDF) is a similar attempt at an application independent data storage framework. However, these methods do address how to design filters and visualization techniques to compress and display a dataset in an intelligent manner.

Other simplification techniques have been used. For example, *filtering* is the process of "cleaning up" arbitrary data. A user gives a criteria for removal of certain data. Those data elements that meet the criteria are removed from the dataset. This type of simplification is usually done post data acquisition stage, where data has been acquired from sensors and user wants to remove *noisy* data. *Smoothing* can also be used to remove noisy data — this is comparable to blurring the data like you would blur an image. And *projecting* data into a lower dimension can be used to curb the dimensionality of a dataset.

2.4 Future Directions

There is still considerable work left in the area of data management and data simplification. At best, these two fields are in their infancy — even younger than the notion of visualization systems. A number of panels have acknowledge the growing problem of data explosion and the express a absolute necessity to curb this growing problem, to propel advances in the sciences [SR98, vDFL$^+$00, MDB87, BGK$^+$88, Ros94]. This is a growing problem that cannot be ignored.

More importantly, these panel members stress that more general methods need to be developed rather than the previous, ad hoc, attempts. Too often techniques developed are too specific to a certain environment (i.e. they do not carry well into other type of applications). Moreover, they realize that not everyone has access to the most advanced visualization systems, and, therefore, it is important that these general solutions offer scalability at a low cost. Suggesting that scientific visualization should be a inclusive field that encompasses the entire scientific community, where its reach is not limited.

Additionally, as Cox illustrates [CE97], the areas of data management and data simplification should not stand on their own. In other words, a good data management technique should not be left without a suitable data simplification algorithm. For instance, even a simplified dataset can be so enormous that it may not fit on a single disk. Therefore, it becomes essential that data simplification and data management become tightly coupled.

To address issues of data explosion and the need for more scalable and general solutions, we introduce a data simplification system that is based on the previous work in computer graphics. Specifically, we take advantage of the large base of mesh simplification research in computer graphics. Much of the work in computer graphics have been general attempts at a common problem in graphics, large data models. Moreover, we introduce principal components analysis and locking to lower the dimensionality of a dataset and fixate important elements, respectively. Our approach is a general system and can be applied in a variety of applications. The next few chapters will introduce the concepts and constructs of our system.

Chapter 3

Geometric Models and Mesh Simplification

The ability to represent geometric objects are essential to the underpinnings of computer graphics. Simple objects, such as a sphere or a cube, can be represented in their exact mathematical form. Other, more complex, objects cannot be represented precisely in their mathematical form. Instead an approximation to the original model is made such that the geometry and topology of the approximation closely matches the geometry and topology of the original, solid form, model. The extent at which two models match is usually based on some visually perceived closeness, i.e. the closer the approximation looks, compared to the original when displayed on the screen, the better the approximation.

Computer Aided Drawing (CAD) systems, must rely on higher degree of precision than the simple "visual closeness" test. Here we must realize that the approximated object may some day exist in a real world, physical, form instead of a conceptual, computer-based, representation. The approximation here is driven by machine precision, i.e. how close a machine, or the manufacturing process, can physically represent the desired object. In part we want $f_o(\mathbf{x})$, the function that describes the original model, and $f_a(\mathbf{x})$, the function that describes the approximation, to satisfy

$$||f_o(\mathbf{x}) - f_a(\mathbf{x})|| < \sigma \tag{3.0.1}$$

where σ is the error introduced in the manufacturing process.

Now the means by which models are represented varies greatly, but the most common,

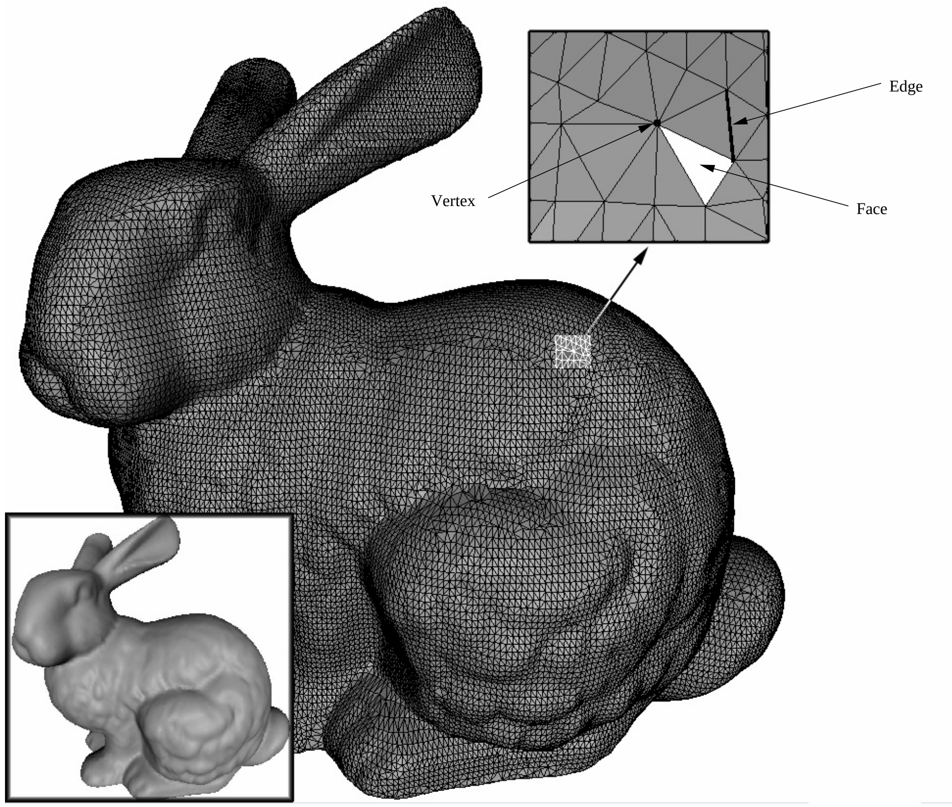


Figure 3.1: Rendered and polygonal mesh

and pervasive, is a polygonal mesh [Wat00].[1] Piecewise linearly connected polygons make up a polygonal mesh and are composed of three fundamental constructs [FvDFH96]:

- *Vertices*: Points in three or two dimensional space.
- *Edges*: Connections between those vertices.
- *Faces*: The plane that is defined by a set of points and edges.

Figure 3.1 illustrates a typical mesh and each of the fundamental constructs. In the figure, note the number of polygonal facets that are involved in the composition of the model. Returning to Equation (3.0.1), we note that the more polygons we add to the mesh the smaller σ will be, only approaching zero. It would only seem appropriate to make the mesh as dense as possible because we would want the most accurate possible representation.

[1] Polygonal meshes are commonly favored because of their simplicity, in representation, and hardware that is commonly available to render them.

Unfortunately, this is a pipe dream, and, on a particular system, we are working in a finite space with limited resources. In fact, the biggest drawback to using a polygonal mesh representation is the number of polygons present in a complex model, hundreds of thousands is not unheard of. For example, recent advances in range scanning technology has allowed the researchers at Stanford University to take three dimensional scans of Michelangelo's David [LPC$^+$00]. The final capture contained over a billion polygons, far more than any generic system can render in real time.

Researchers are well aware of the problem of large meshes. Even in the early days of computer graphics, James Clark saw the need to reduce the size of polygonal objects [Wat00]. But it was not until the early 1990's that any automatic techniques were developed to deal with the large polygonal models. These methods that were developed used decimation/simplification techniques that iteratively removed vertices until a bounded error, like that in Equation (3.0.1), was achieved.

3.1 Notation

Before we review mesh simplification, we will introduce some notation that will be common throughout this paper. We will adhere to notation that can be found commonly in any mathematics textbook.

First, all column vectors and matrices will be represented in bold letters. Specifically, matrices will be presented in boldface uppercase letters and vectors will be expressed in boldface lowercase letters. Single dimensional variables will be presented in lowercase, non-bold font, and sets will be represented in uppercase, non-bold font. Moreover, all variables will be real numbers, and all vectors and matrices will be n dimensional, unless otherwise specified. For example, $\mathbf{X}$ is an $n \times n$ matrix; $\mathbf{x}$ is a n dimensional vector; X is a set; and x is a real number.

Non-quadratic functionals will be presented in lowercase characters, and quadratic and higher degree functionals will be bold, uppercase characters. All functional variables will be explicitly differentiated from matrices and single dimensional variables.

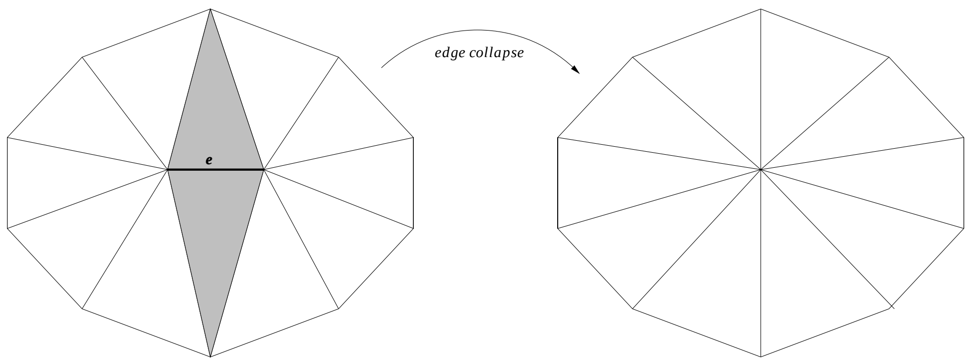
3.2 Mesh Simplification

Polygonal mesh simplification has been an active area of research in computer graphics for the past several years. The earliest work focused on preserving the topology of the mesh. These methods developed use a metric, $m : R^3 \rightarrow R$, to measure the amount of error, $\epsilon \in R$, in a sample set, D. Each element in the sample set consists of a data element, $\mathbf{d} \in R^3$, which is simply a spatial position, i.e. vertex. Interestingly, all mesh simplification use this criteria to evaluate a mesh, or

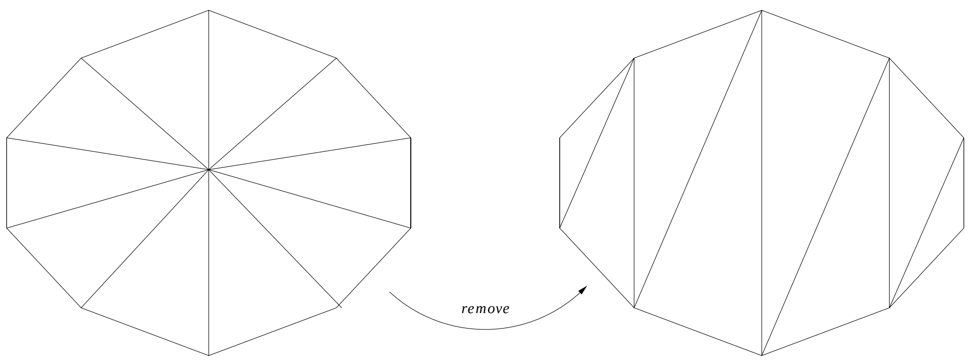
$$m(\mathbf{d}) = \epsilon. \tag{3.2.1}$$

Those regions that contributed little error can be removed or decimated from the mesh. After this removal, the hole must be filled. Here a re-triangulation method, like [Tur92, SZL92], is used to fill in any holes that may be caused by a removal, see Figure 3.2(b). Later work discovered that it was not necessary re-triangulate a region [GH97]. Instead, we can collapse an edge into a point, see Figure 3.2(a). In this instance, ϵ is some combination of the vertices defining the edge. The point of collapse in Figure 3.2(a) is usually the point where ϵ is the smallest.

Now the metric m varies greatly across the literature of mesh simplification. Early work by Rossignac and Borrel [RB93] clustered groups of vertices into a single vertex and used the coarser representation when the vertex is further away. They later discovered that this did not produce accurate results because there was no way to mathematically express the error that was induced through this process. So Ronfard and Rossignac [RR96] derived a new method that used the sum of squared distances to the planes surrounding a given vertex. Garland and Heckbert [GH97] later improved this method by encoding the sum of squared distances in a quadric error metric, see Section 3.2.1. Lindstrom and Turk [LT98] followed up with a similar approach using a counter intuitive memory efficient model. Schroeder et al. [SZL92] defined an average plane spanned by a local set of vertices and measured the distance from the average plane to other points. Others like Cohen et al. and Hoppe et al. [CVM+96, HDD+93] take a different approach. Cohen uses envelopes that surround the original mesh by a given user defined threshold. Simplification occurs until the decimated mesh seeps outside of the envelope (with Cohen's method any simplification algorithm can be used — it is just another particular way to bound error). Hoppe, on the other hand, solves a modified least squares problem to properly fit a mesh with a given set of vertices.



(a) Edge Collapse.



(b) Re-triangulation after vertex removal.

Figure 3.2: Methods of vertex removal.

3.2.1 Quadric Based Simplification

In our implementation we use two quadric based simplification techniques. As a precursor, we will provide some background in this area. Also, this will provide some intuition of a typical metric.

Quadric based simplification has recently received a great deal attention from many people in the computer graphics community. Initially, Garland [GH97] introduced the notion that a quadric based metric could be used to measure ϵ on a geometric mesh without surface attributes. Specifically, his metric measures the squared distance from $\mathbf{v}$ to a plane, where our plane is defined by a normal vector, $\mathbf{n}$, and a point on the plane, $\mathbf{p}$ (on a triangle mesh we can construct this plane using a triangle face, f). So, $\mathbf{n}^T\mathbf{p}$ defines the plane containing

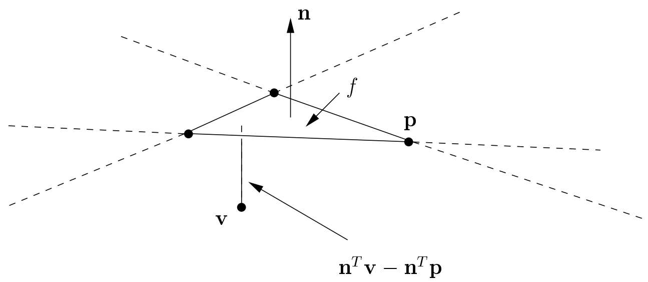


Figure 3.3: The distance between two planes.

$\mathbf{p}$, and $k = \mathbf{n}^T\mathbf{v}$ defines the parallel plane containing $\mathbf{v}$. Figure 3.3 illustrates that given a normal, $\mathbf{n}$, and two points, $\mathbf{v}$ and $\mathbf{p}$, we can find the distance between these two points by letting $\mathbf{v}$ lie on a plane parallel to $\mathbf{p}$. Thus our distance squared function for this face, f, from $\mathbf{v}$ to $\mathbf{p} \in f$ is simply

$$\mathbf{Q}^f(\mathbf{v}) = \left(\mathbf{n}^T\mathbf{v} - \mathbf{n}^T\mathbf{p}\right)^2 \tag{3.2.2}$$

This is essentially a quadratic polynomial and can be expressed in matrix form, also known as a quadric. Hoppe and Garland [Hop99, GH97] show that (3.2.2) is equivalent to

$$\mathbf{Q}^f(\mathbf{v}) = \mathbf{v}^T\mathbf{A}\mathbf{v} + 2\mathbf{b}\mathbf{v}^T + c \tag{3.2.3}$$

where $\mathbf{A} = \mathbf{n}\mathbf{n}^T$, the vector $\mathbf{b} = k\mathbf{n}^T$, and $c = k^2$. Therefore, given a $\mathbf{v}$ we can evaluate error for $\mathbf{v}$ by simply applying (3.2.3). We only need to store $(\mathbf{A}, \mathbf{b}, c)$ as our *fundamental quadric.*

However, before this can take place we must first assign a quadric for every $\mathbf{v} \in D$. Naturally more than one quadric can be assigned per vertex — recall that a vertex can share many faces (Figure 3.1) — so we take the sum of the quadrics. This gives us

$$\mathbf{Q}^v = \sum_{f \ni v} area(f)\,\mathbf{Q}_f \tag{3.2.4}$$

One can visualize (3.2.4) as creating a new plane in space, like that in Figure 3.4. We weight the quadric by the area because we do not want large triangles with little error to have more significance than small triangles with nearly equal error.

Figure 3.4 shows a cross section of a simple mesh with only two faces, marked with a dark lines. Each face forms a plane that extends in the positive and negative zeta, which has

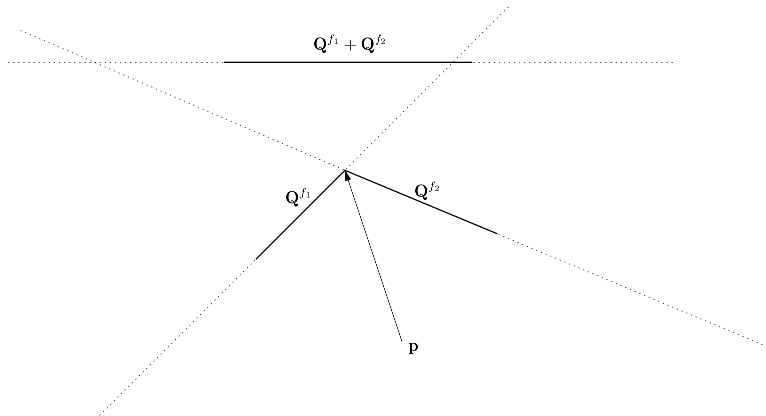


Figure 3.4: Sum of fundamental quadrics.

been projected down onto a two dimensional plane. Therefore, we have infinitely long lines, dotted, that represent the planes that are spanned by the faces f_1 and f_2. Here f_1 and f_2 share a common vertex $\mathbf{v}$, and we can build two fundamental quadrics, $\mathbf{Q}^{f_1}$ and $\mathbf{Q}^{f_2}$. Now we do not want to store n different quadrics for the n faces that may share a vertex, and, fortunately, we do not have to. We can simply add fundamental quadrics together. Figure 3.4 illustrates a possible new plane that is formed by adding two fundamental quadrics, labeled $\mathbf{Q}^{f_1} + \mathbf{Q}^{f_2}$. In addition, quadratics are vector spaces and therefore adhere to other mathematical properties, like subtraction and scalar multiplication [LE96].

Once we have completed this pre-computation step, we can then perform our mesh decimation by simply evaluating

$$\mathbf{Q}^{v}(\mathbf{v}) = \epsilon \tag{3.2.5}$$

for each vertex on the mesh. We contract those edges, $e = (\mathbf{v}_1, \mathbf{v}_2)$, in the mesh that have vertices, $\mathbf{v}_1$ and $\mathbf{v}_2$ with a "small" ϵ. After the collapse we have a new vertex, $\mathbf{v}_{new}$, with the fundamental quadric $\mathbf{Q}^{\mathbf{V}_1} + \mathbf{Q}^{\mathbf{V}_2}$.

One interesting problem in quadric based error is how to determine $\mathbf{v}_{new}$, the point to collapse an edge to. A simple solution would be to collapse the edge to the midpoint or end points of a edge, that is we choose a $\mathbf{v}_{new}$ such that

$$\mathbf{v}_{new} \in \left\{ \mathbf{v}_1, \mathbf{v}_2, \frac{\mathbf{v}_1 + \mathbf{v}_2}{2} \right\}.$$

The problem with selecting $\mathbf{v}_{new}$ via the midpoint or end points is that we are not guaranteed that $\mathbf{v}_{new}$ is the point of absolute minimum error. In other words, the point that induces the least amount of error on the mesh. Sure, the edge e has the smallest error of all edges in the mesh, but the absolute minimum error is in the neighborhood of e. This minimum will occur when the first derivative is equal to zero [GH97], or

$$\mathbf{A}\mathbf{v}_{new} = -\mathbf{b}. \tag{3.2.6}$$

All we need to do is solve the system to find the appropriate $\mathbf{v}_{new}$.

3.2.2 Simplification with Attribute Information

Initial work in simplification was only to preserve geometric qualities of an arbitrary mesh, but it was soon realized that geometric simplification performs poorly on objects with additional features. For example, Figure 3.5(a) demonstrates an image map that has been placed over a mesh.

Each vertex is assigned a color value from the image map. Now if we applied just a geometric only based simplification algorithm, then all attribute information would be lost, as in Figure 3.5(b) — notice how the entire image plane has been distorted by geometric only simplification. If we calculate our error with attribute information in mind, we can preserve attribute information, and the entire structure of the mesh, as in Figure 3.5(c).

Besides color information, geometric meshes can encode additional data at their vertices. The most commonly included attributes are normals and texture information. Normals are used to "explain" how light interacts with the mesh (these normals can be different from the normals described in quadric simplification). Using normals properly can make an object, that is made up of polygonal facets, look smooth (see inset in Figure 3.1). Texture information map a, usually repeatable, pattern to a mesh. This is to make objects "look" like real world objects, because real world objects are rarely solid in texture. Using each of these properties correctly, along with proper lighting, can produce strikingly realistic images. But large meshes with high polygon count can decrease system performance, and a simple geometric based simplification algorithm may not preserve some of the above features properly. Thus, great attention has been focused on preserving features, such as normals, colors, and textures, while preserving the objects geometry [COM98, GH98, Hop99, Hop96, BS96]

In general, these *appearance preserving* simplification methods use a metric, $m : R^{3+n} \rightarrow R$, to measure the amount of error, $\epsilon \in R$, in a sample set, D. Each element in the sample set consists of a data element, $\mathbf{d} \in R^{3+n}$, that contains a geometric location, $\mathbf{v} \in R^3$, and attributes, $\mathbf{a} \in R^n$, taken at that geometric position. Here we have a similar approach as in (3.2.1). Those areas that contribute little error can be removed or decimated from the mesh. The end result is a much coarser mesh that requires less storage yet still represents key features of the original mesh.

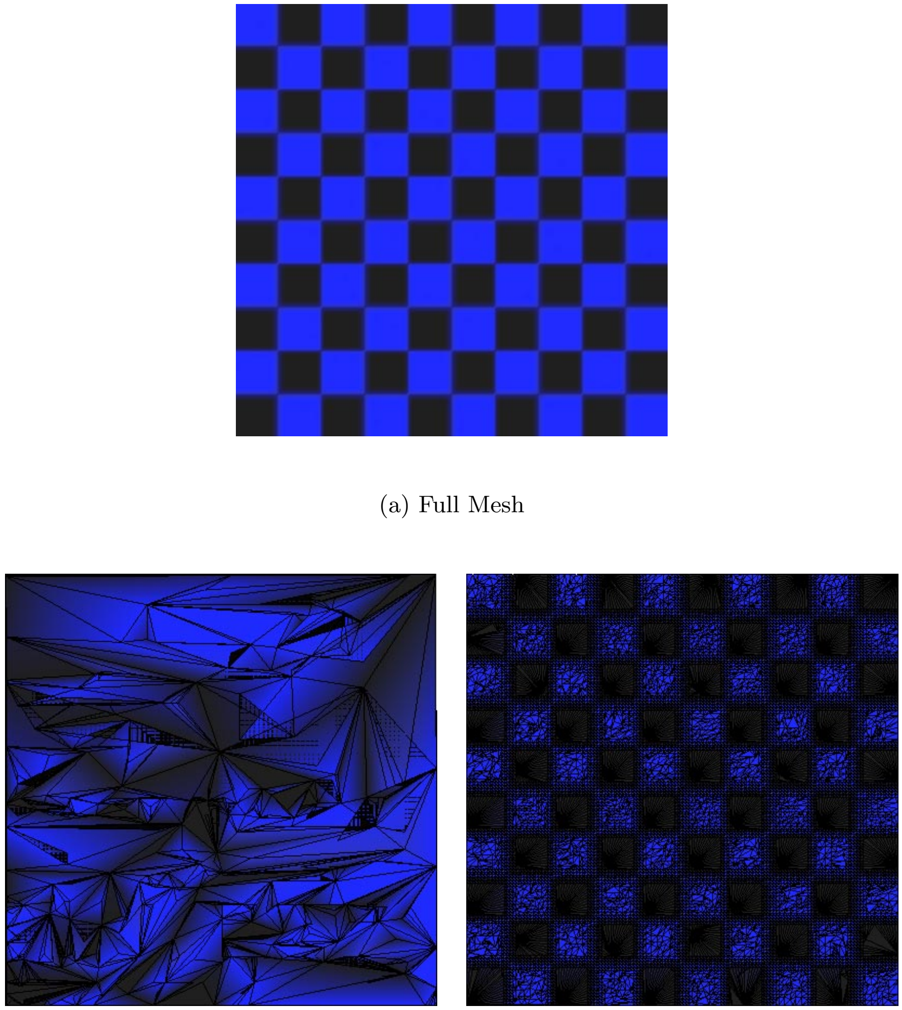
Now this measurement m varies from method to method. Bajaj [BS96] uses a signed distance function for m, where m measures the change in topology from mesh M_i to M_{i+1} after $\mathbf{d}$'s removal. The removal of $\mathbf{d}$ results in a new triangulation in the region that contained $\mathbf{d}$. The original mesh makes a projection onto this new triangulation, and attribute error is measured by taking the deviation between attribute values interpolated along the new mesh and attribute values interpolated along the old triangulation. If this deviation satisfies a user given error constraint the triangulation is kept and the process is repeated.

Cohen et al. [COM98] use a decoupled representation to attain appearance preservation during simplification. They create a surface parameterization that is used to map attributes onto an attribute plane. Typically, this parameterization is given, e.g. normal and texture maps, but in a visualization environment these maps need to be created automatically using a technique similar to [LSS$^+$98, EDD$^+$95]. Regardless of how the parameterization is attained, we have these attribute planes, p_{a_i}, which map attribute information onto the geometric mesh. Their algorithm simplifies a given geometry by removing those edges that cause no or negligible distortion in the parametric space, texture space. Then after a simplified geometry has been constructed attribute planes are mapped back onto the surface.

An entirely different approach taken in mesh simplification is multi-resolution analysis where wavelets are used to decompose a mesh into a base mesh with varying levels of granularity. Originally, multi-resolution analysis was performed on mesh without attribute information [EDD$^+$95], but Certain and others [CPD$^+$96] include color in their filterbank analysis. This method could be generalized to include other attribute information.

3.2.3 Attribute Simplification with an Energy Function

Hoppe extends his earlier work [HDD$^+$93] and defines an energy function composed of several terms. This energy function is used in two ways. First, it is used to measure the error from our original mesh to our optimized mesh. Second, he uses his energy function to



(a) Full Mesh

(b) Geometry Only

(c) Attribute Simplification

Figure 3.5: Mesh simplification with attribute information.

define a series of optimization problems where ϵ is made as small as possible. He specifically addresses simplification with $\mathbf{a}$ and includes an additional metric, m_a, where m_a is simply the sum of distances from $\mathbf{a}$ to the parameterized mesh. A minimized value is then found by solving a least squares problem.

Our implementation of this metric is a simplified version. Specifically, we do not perform the least squares fit to the mesh with the new mesh. Instead we evaluate the estimated energy imposed from an edge collapse. We restrict to testing collapses to the end points and the midpoint of the edge, like that suggested by Hoppe. That which has the lowest cost is placed into the priority queue among other candidate edge collapses. Like above, the edge is collapsed when it is removed from the queue.

Energy, m, is calculated using the sum of three functions. Specifically, two functions are used to measure the surface error, and the final function measures error in attribute space.[2] Our function breaks down into the following form:

$$m = m_{dist} + m_{spring} + m_{attribute}. \tag{3.2.7}$$

Without going into too much detail we will briefly mention how each component breaks down.

The geometry of the original mesh is recorded by copying the original set of points from the original dataset, $\mathbf{X}$. As Hoppe describes, this records the geometry of the old mesh. This "recording" is necessary because we want to be able to measure the distance from the new, simplified mesh, to this old mesh. Now the m_{dist} term is found by taking the projection of each of the points in $\mathbf{X}$ onto the new mesh. The energy is the distance of this projected points to the original points, $\mathbf{X}$. The m_{spring} term is the sum of edge lengths. The final term, $m_{attribute}$, is the measurement of attribute distance from the point of projection on the new mesh. Details of these metrics can be found in Hoppe's paper [Hop96].

3.2.4 Attribute Simplification with Quadrics

The quadric error metric that was developed by Garland and Heckbert [GH97] was shown to produce fast and accurate geometric based simplification. Later, they extended their work to work with surface attributes [GH98]. Essentially, they generalized their previous method of measuring the squared distance to a plane, to measuring the squared distance to the hyperplane in $\mathbf{R}^{3+n}$. As a result of this extension, the parameters and variables in Equation (3.2.5) are vectors and matrices in R^{3+n}. The evaluation of the metric remains the same, and we simply supply a data element $\mathbf{d} \in R^{3+n}$.

It was later shown by Hoppe [Hop99] that this generalization may result in a misrepresentation of the actual error. He realized that to capture how an attribute changes over a surface we must measure the change on that surface and not some point in n-dimensional space. What he sought was an equation similar to (3.2.3). In other words, we need to find

[2] Hoppe also includes another function that measures attribute discontinuity. However, we exclude this function because our fields are all scalar and have no discontinuous values.

the distance between attribute values at a point on the face of the polygon to the point in space. Figure 3.6 illustrates this measure. Here we have three attribute values at each of the vertex positions of the face (s^{v_1}, s^{v_2}, and s^{v_3}), an attribute value in space (s_d), and an interpolated value on the face of the triangle (s_p) that we are using to measure the difference in attribute values, $(s_d - s_p)^2$. To find s_p we need to know how s^{v_1}, s^{v_2}, and s^{v_3} change over the face of the triangle. Thus we need to create some functional $\hat{s}(\mathbf{p})$ that gives us the attribute value at the position $\mathbf{p}$. More specifically, we need a functional for every $s_j \in \mathbf{a}$. This functional is easily derived since our scalar values change linearly over the face of the triangle [Hop99]. Or,

$$\hat{s}_j(\mathbf{p}) = {\mathbf{g}_j}^T \mathbf{p} + d_j$$

where $\mathbf{g}_j$ explains how attributes $s_j^{v_1}$, $s_j^{v_2}$, and $s_j^{v_3}$ change over a given face, and d_j is the initial value for $\mathbf{g}_j$. However, $\mathbf{g}_j$ and d_j are unknown, and we have to find them. To do so we first note,

$$s_j^{v_1} = \mathbf{g}_j^T \mathbf{v}_1 + d_j$$
$$s_j^{v_2} = \mathbf{g}_j^T \mathbf{v}_2 + d_j$$
$$s_j^{v_3} = \mathbf{g}_j^T \mathbf{v}_3 + d_j$$

Moreover, $\mathbf{n}$ will be orthonormal to $\mathbf{g}_j$ and $\mathbf{n}^T \mathbf{g}_j = \mathbf{0}$. Thus we have a system of linear equations and we can solve to find $\mathbf{g}_j$ and d_j.

So now our distance measure, for attribute deviation, is simply

$$\mathbf{Q}_j^s = ({\mathbf{g}_j}^T \mathbf{p} + d_j - s_j)^2.$$

And we can construct a new quadric based on the sum of attribute deviations such that

$$\hat{\mathbf{Q}}^f(\mathbf{d}) = \mathbf{Q}^f(\mathbf{d}) + \sum_{j=1}^{m} \mathbf{Q}_j^s(\mathbf{d}). \tag{3.2.8}$$

Hoppe shows that this quadric breaks down into the same fundamental form as in (3.2.4). Hence we have

$$\left(\mathbf{A} = \left(\begin{array}{c|c} \mathbf{n}\mathbf{n}^T + \sum \mathbf{g}\mathbf{g}^T & -\mathbf{g}_1 \cdots - \mathbf{g}_n \\ \hline -{\mathbf{g}_1}^T & \\ \vdots & \mathbf{I} \\ -{\mathbf{g}_n}^T & \end{array} \right), \mathbf{b} = \left(\begin{array}{c} k\mathbf{n} + \sum d_j \mathbf{g}_j \\ \hline d_1 \\ \vdots \\ d_n \end{array} \right), c = k^2 + \sum {d_j}^2 \right)$$

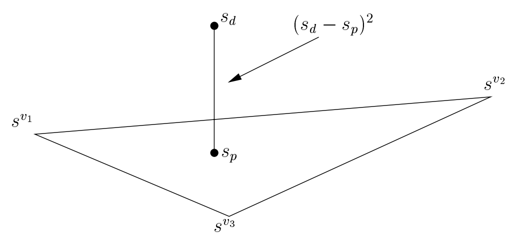


Figure 3.6: Distance between attribute values on a polygonal face.

where k and $\mathbf{n}$ are from Garland's QEM (Quadric Error Metric) that was described earlier.

The same rules apply as did before — find all of the fundamental quadrics and iteratively collapse lowest cost edges from the mesh. But we now have a more accurate means of measuring error than before. And despite the similarities between these methods, Hoppe versus Garland, the metrics provided are fundamentally different. Hoppe evaluates attribute error on a layer by layer basis, and Garland evaluates attribute error collectively. Generally, regardless of the type of simplification algorithm, simplification depends on vertex positions and the attributes attached to those positions, and error is measured by some combination of vertex positions and attribute information. This observation is keen to using the previous work in mesh simplification as a foundation for our problem of simplifying large multi-dimensional datasets.

In the upcoming chapters, we will present our work in mesh simplification. Specifically, its application in a data-based environment. We will look at the similarities and dissimilarities between model-based data and arbitrary scientific data, as well as, the problems of using mesh simplification in a data simplification environment. Principal components analysis will be introduced as a general extension to any existing mesh simplification algorithm, and we will introduce spatial fixation to control movement of vertices.

Chapter 4

Data Simplification

It can be easily seen, by review of the literature, that mesh simplification is a very active and diverse area of computer graphics. Interestingly, the problem of mesh simplification does not deviate far from its counterpart in scientific visualization. That is, in scientific visualization we have a very dense data model that contains a high amount of "redundant" data. For instance, there may be regions of data that are planar, i.e. have common attribute values. Therefore, if we can remove all of the similar data elements from those regions and leave a few data elements as representatives of that region, then we can eliminate redundant information from the dataset and reduce its size. In a way, we seek to fit the original data model with a new data model that "fits" the original data model as closely as possible. Or mathematically,

$$||f_{D_o} - f_{D_n}|| < \sigma \tag{4.0.9}$$

where f_{D_o} is the function that describes the original data and f_{D_n} is the new, simplified data, and σ is how close we want to get to the original. Depending on how much redundant information present will dictate how much data will be removed with respect to σ. In other words, there will be more simplification in large areas that share similar properties.

Now as we saw in the previous chapter, geometric models often present the same problem. That is, we have a dense model that we wish to fit with a less dense model, yet still accurately "fit" the old, denser model. In fact, mathematically, the two problems can be identically expressed. And, practically, we only need to create a suitable "link" between these to mathematically similar problems. A bridge, per se, that allows us to take advantage of this similar problem. However, to justify our "hunch" we must ask the following before

Geometric Models	Visualization Datasets
Samples take over a finite space connected by a graph like structure, i.e. a mesh.	Samples taken over a domain, can be 3-space, but it is not limited to 3-space, e.g. samples taken over time. Also, the samples can be associated with a mesh [CE97].
Attributes are typically scalar but can be discrete, e.g. colors.	Mostly scalar data, but can have discrete data values, e.g. survey with "yes" or "no" answers.
Small, finite number of attributes per data element, i.e. color, normals, and texture.	Can have an arbitrary number of attributes per data element.
Attribute values are low precision and usually vary between $[0, 1]$.	Attributes can be arbitrary precision (typically 8 byte doubles [SR98]).
Exact spatial location is not important. It is only necessary that the topology be preserved.	Spatial location may be important, i.e. the location of a weather station

Table 4.1: Similarities between geometric meshes and scientific data.

developing any approach:

- What issues arise when one tries to apply mesh simplification to simplify an arbitrary dataset?
- Can we simply use a mesh simplification algorithm verbatim on a visualization dataset?
- Are new algorithms necessary to simplify scientific datasets?

To address these posed questions we have outlined the important properties that are necessary in mesh simplification algorithms, and we have mapped these properties into their data simplification equivalent. The table above is this mapping, Table 4.1. Making note of this table, can we apply mesh simplification verbatim? Succinctly, No. Despite the many commonalities that are shared between geometric meshes and visualization datasets, we cannot simply apply mesh simplification verbatim because

1. Differences in relational representation, i.e. geometric objects are spatially related, where datasets may be related in other ways.
2. Precision issues in the mesh simplification algorithm. Simplification algorithms must support higher-order precision.

3. Spatial location of data points are important. Moreover, we do not want large amounts of simplification in large planar regions because the loss of information in these regions may confuse the user.

4. Most mesh simplification algorithms handle, or designed to handle, only a minimal attribute set. For our concerns, it is important that a mesh simplification algorithm handle the next generation of datasets with more than 500 attributes per data element [SR98].

Of these four issues, only two warrant new extensions to the previous mesh simplification work. The other two, (1) and (2), are simply changes in representation; therefore, they do not deviate far from the design requirements of a particular mesh simplification algorithm. For instance, datasets may have different relational properties, e.g. growth over time, percentage yield over time, etc. However, conceptual representation of these different relational models typically follow spatial properties, e.g. assign time the *x-axis* and assign growth to be the *y-axis*. As these two parameters vary a mesh is formed. In cases where a temporal component exists in conjunction with spatial properties, we can simply perform simplification at each time step. Typically, changes over time are often represented as an animation [HS90]; therefore, it would not be unreasonable to use this representation.

As it turns out, case (2) also holds little relevance. As we saw in the previous chapter, most algorithms are based on geometric relations that have mathematical properties used to generate an appropriate ϵ, e.g. distance to a plane. Therefore, precision is only a matter of having the appropriate hardware that can handle the higher precision requirements. In other words, the same mathematical properties will hold. Only the number of numbers that can be represented will change

The following sections will address the remaining two cases, (3) and (4). In these two sections we will demonstrate how to curb the dimensionality of a dataset and fixate important data positions on the mesh. First we will review *locking* to fixate data elements to a given spatial location. Then we will explain principal components analysis and how it can be used to reduce the dimensionality of a dataset.

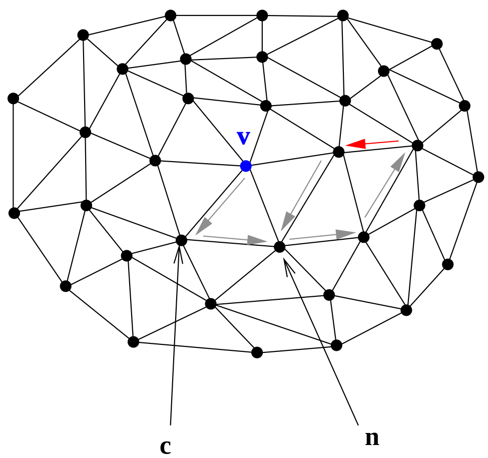
4.1 Fixating Spatial Positions

As we have already noted, position may be important in a data model. Particularly because we are assuming a mesh has significance with regards to a collection of data elements; however, this may not be the case. For instance, perceptual textures [HE98] assign texture properties at each data element. There is no underlying mesh assumed, and, thus, the mesh is not important. Now suppose that the dataset contains a high amount of redundancy; this implies that we have large areas of no or little variability. The regions of high redundancy would be removed by a specific algorithm. However, in the final visualized model this could create undesirable artifacts. Figure 4.1(b) demonstrates this planing that could occur during simplification — for illustrative purposes we exaggerated the amount of simplification. In this figure, we see those data elements that remain after simplification of a fully populated dataset. The arrows indicate regions that may be undesirable or unwanted, i.e. the regions that have no information present. To limit this *planing* we introduce *locks* in our implementation.

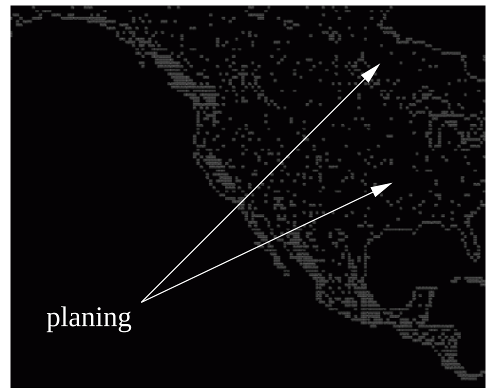
Locks simply fix a data element to its given location. Internally, we can allocate one bit of information that indicates if the data element lock is on or off. Naturally, this representation limits the amount of simplification that can be performed on the mesh. However, *locks* on the mesh will reduce undesirable *planing* artifacts. That is, if we lock an element that element cannot move and, therefore, planing will not occur.

To set the locks we use a simple depth first search to determine if a lock can be "turned on." Specifically, a $\mathbf{d}$ can be locked if no other locks in the Δ-*neighborhood* of that data element are already locked, e.g. a 3-neighborhood would be those vertices that are separated by 3 edges from the queried vertex. A similar method would use *thresholding* where $\mathbf{d}$ is marked whenever it is involved in an edge collapse. If the data element is marked s times, a given threshold value, then it cannot be removed after that point. The difference between the two methods is that one requires an additional precomputation work, and the latter can be done during simplification. Additionally, thresholding requires slightly more space overhead than locking (depending on s), where locking only requires a bit of information.

There are some additional points to make about the depth first strategy used to perform uniform locking. Specifically, a brute force exhaustive search should be avoided. Instead, while searching, the search should remember what vertices it has visited. This prevents loops from forming in the search. Figure 4.1(a) illustrates this point. The arrows indicate



(a) Searching without remembering



(b) Planing

Figure 4.1: Locking

a potential path taken by the depth first search as it traverses the mesh looking for locked vertices in the Δ-neighborhood. Since none of the vertices are locked in this example, $\mathbf{v}$ should be locked. But we need to find this out as quickly as possible without exhaustively searching all the possible paths from $\mathbf{v}$ to Δ edges away.

So in addition to remembering where we already visited, the search can be guided by a *proximity function*. In other words, the edges of the mesh can take us further away from $\mathbf{v}$, in Figure 4.1(a), or move us closer to $\mathbf{v}$. The paths that move us away from $\mathbf{v}$ are the ones we are interested in. Particularly because these places are where other vertices may be locked; thus preventing us from locking $\mathbf{v}$. So we can move from $\mathbf{c}$ to $\mathbf{n}$, as in Figure 4.1(a) if

$$||\mathbf{v} - \mathbf{c}||^2 < ||\mathbf{v} - \mathbf{n}||^2. \tag{4.1.1}$$

Thus, we can avoid those situations that do not move us towards our goal, depicted by the red arrow in Figure 4.1(a). Moreover, we can use this *proximity function* as an alternative to the Δ-neighborhood classification, where a vertex is permitted to be locked if no vertices within some radius of $\mathbf{v}$ are locked. This allows lock searching to be used on meshes that do not have uniform spatial distribution, like that in Figure 4.2. In this example, a 2-neighborhood search would lock the blue colored vertex because there is not a locked, in red, vertex within 2 edge links of the queried, blue, vertex. Of course, this may not be

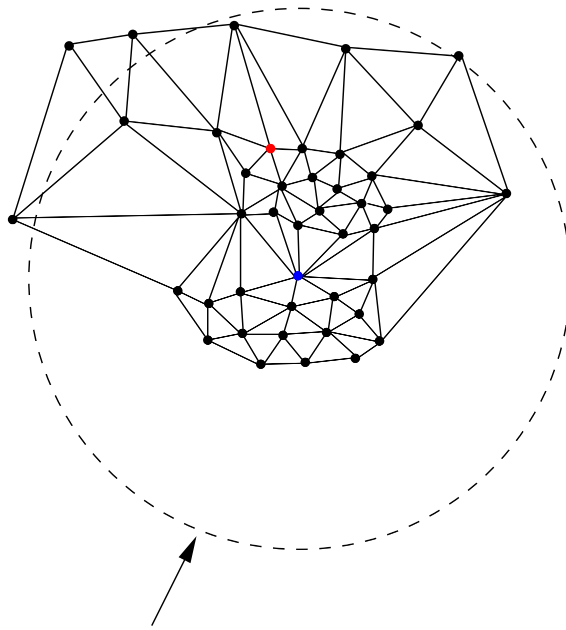


Figure 4.2: Non-uniform spatial distribution locking

what the user desired. Instead he may want a more uniform locking across the mesh. In the non-uniform case, the actual goal is to test that the positional part of a data element is not locked within a certain radius of the blue colored data element. So in addition to our guidance equation, we need to add another test. Specifically, a vertex, **v** can be locked if

$$\forall\, \mathbf{p} \in \{\mathbf{d} \in D \;\mid\; ||\mathbf{v} - \mathbf{x}|| < r\} \tag{4.1.2}$$

p is not already locked. Where **x** is the positional part of a data element **d**, and r is the desired radius. This extra component can be added to the depth first search process as a stopping condition.

Whenever a locked vertex is encountered during simplification, the vertex is not permitted to move. So either the vertex is skipped because it is locked or the edge associated with the vertex is collapsed to the locked vertex. Naturally, the point after collapse inherits the locked properties. In the case of an edge collapse, if both vertices of the edge are locked then edge collapse is thrown away and a new edge is obtained from the queue of available edges.

In addition to turning on a lock flag, we add a constant to the error associated with lock vertex, see Figure 4.1. Adding this additional constant prevents the algorithm from picking

```
evaluate_locked_data_element_costs( )
{
  foreach v in data_elements do {
    if ( element_is_locked( v ) ) {
      cost_of_element[ v ] = evaluate_cost( v ) + locking_constant;
    }
  }
}
```

Figure 4.3: Adding additional costs to locked elements.

a vertex that contains a locked vertex. Recall that simplification algorithms pick vertices to remove based on their cost. The higher the cost of removing the less likely the vertex is removed. The choice of constant can vary across implementations, and it depends strictly on how "hard" you want to lock **d**. The harder the lock, the larger the constant, and the less likely a locked vertex will be picked for decimation. In our implementation the constant is a user defined value and it is multiplied to the error on the locked vertex. Accordingly, this allows the user to decide how likely locked vertices are evaluated in the algorithm. It should be noted, however, that adding this cost is not necessary for locking. We only added this extension to prevent the simplification algorithm from picking the lock vertex.

As we suggested, the notion of *locks* have additional benefits in a visualization environment; that is, it may be important that some data elements not be moved/removed. For example, gridded weather data is formed from data collecting weather stations.[1] These stations report the current weather conditions (temperature, rain fall, pressure, etc.) specific to the stations *exact* location. A nearest neighbor interpolation algorithm [FvDFH96] fills in the gaps along a uniform grid. However, these interpolated values are only approximations, and, therefore, may not contain the exact conditions at the interpolated position. Hence, the closest gridded location to the original station is the most accurate and should be kept. Existing work in mesh simplification does not provide means to classify certain locations as significant; therefore, the only way to protect our original data appropriately is by fixating them to their defined positions (when necessary).

[1] Gridded data is spatially uniform.

4.2 Handling High Dimensionality

As we mentioned previously, dimensionality of an arbitrary dataset can pose a potential problem for current appearance preserving mesh simplification algorithms. This is because these algorithms [Hop99, Hop96, HM00, BS96] were only intended for use on a small attribute space. In all cases, error is calculated using a given data element. Specifically, the mesh simplification algorithm uses the given data element and surrounding data elements to calculate the error caused by removal of the element. A particular algorithm must, at the very least, look at some factor times the number of attributes present in the data element. So, the computational cost of evaluating one data element is at least $\Theta(c(n+3))$ or $\Theta(cn)$, which is the lower bound for the cost of evaluating a single point. That is, we have to look at each attribute value to derive some notion of cost. Hence, the only way we can improve the cost of this operation is to approximate the data element with a lower dimensional counterpart.

In lieu of this lower bound, it must be noted that not all algorithms meet this expected lower bound. Depending on the algorithm the computational cost can be from $\Theta(n)$ to $\Theta(n^2)$, and this depends strictly on evaluating error on the mesh. In these cases, most algorithms search for an optimal mesh that fits the old mesh as close as possible. Finding this optimal mesh is much more expensive vis-á-vis evaluating the error on the mesh. For instance, Hoppe [Hop96] uses a least squares fit to find a point, and attribute values, that induces the least amount of error on a mesh. In this case, the computational cost of finding a best fit case is $\Theta(n^2)$, in the general case. Therefore, given an attribute space of 500 elements, the number of operations to find this fit is 250,000; far too costly for a single evaluation. Moreover, in quadric based simplification [GH98], an optimized data element is found by solving a linear system of equations. Solving such a system in via direct methods is an $\Theta(n^3)$ operation. Thus as the dimensionality of a data element becomes large, the cost of performing these "evaluation operations" severely hinders the simplification process.

So, it would be useful to derive a new data element that approximates the old data element such that this new element has a much lower dimensionality. In other words, if we can find a p-dimensional data element that encodes the original n-dimensional data element then the cost of evaluating a data element would be with respect to p versus n. Moreover, if p is significantly smaller than n then that would imply a significant savings during in the simplification process.

Principal components analysis (PCA) is the process of finding the directions of variability in a dataset. Using PCA we can find the directions of largest variation, and, moreover, we can find those directions that account for a specific amount of variation in the dataset. It is those directions of large variability that will effect the error calculated by the mesh simplification algorithm, and, so, the least variant components have little significance in the calculation of error. Therefore, we can give up some of our knowledge of the dataset, the least variant parts, to generate a new data element that is p-dimensional versus n-dimensions.

To reduce space overhead, extra computation, and speed the process of simplification, we introduce the use of principal components analysis. However, we have to realize that by the introduction of principal components analysis we are transforming the data into a set of uncorrelated data. Because the user will not readily be able to understand the data in principal component form, we have to provide a way to perform simplification using the principal components, yet without changing the data itself or adding additional cost in space overhead. The obvious solution, and the one we chose, is to keep the data in its original form, but transform the data into principal component space when performing the actual simplification.

Before we begin our discussion on PCA, it is noteworthy to mention that by transforming the data into principal component space we cannot perform an optimization on the original data. Instead the optimization is with respect to the transformed data. To solve the problem one can either restrict, or not allow, the optimization of a data element, or, as we will show, try to unproject the data element back into original space, i.e. unwind the principal component transformation.

4.2.1 Principal Components Analysis

In general, anytime you try to reduce the dimensionality of a dataset your are bound to lose some information in that process. We want to limit this loss. Principal component analysis (PCA) allows us to find the directions of large and small variability, and it is by looking in the directions of large variability that we can transform our original n dimensions into a hopefully much smaller p.

As explained by Flury [FR88], this works well because often in high-dimensional data we find highly correlated variables. For example, in a full blown weather model we have many attributes, air pressure, precipitation, temperature, and wind-speed. Each of these variables are related to one another, in some way, during period of time the sample was taking. Without this correlation, weather prediction is not possible. In fact, most data will fall under this category, correlated data that we wish to understand. Thus using a method that transforms data from a correlated set to an uncorrelated set will allow us to capture most of the data in a hopefully much smaller dimension.

But how is this done? How do we find the corresponding uncorrelated data given correlated data? The answer to this question, and the key to principal components analysis, is what is know as the principal axes theorem.

4.2.2 Principal Axes Theorem and Eigenanalysis

The principal axes theorem states[2] that "any quadratic form in n variables $\mathbf{Q}(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x}$ is equivalent by means of an orthogonal matrix $\mathbf{Z}$ to a quadratic form

$$\hat{\mathbf{Q}}(\mathbf{y}) = \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2$$

where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad , \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} ."$$

This is best illustrated by an example. Figure 4.4 illustrates two ellipses that are identical in shape, but defined differently in terms of there quadratic form. Specifically, if we aligned y_1 and y_2 with x_1 and x_2, respectively, the resulting figures would look identical. Moreover, the left-most figure is in standard form, i.e. we could write the equation of that figure if we knew its radii in the x_1 and x_2 directions. The right-most figure, on the other hand, is not in standard form and has a x_1x_2 coefficient that describes how the ellipse varies on a non-principal axis in the coordinate system of x_1 and x_2 (let $\mathbf{A}$ be the matrix of this quadratic form). However, with respect to the coordinate system of the right-most ellipse, y_1 and y_2, there is no y_1y_2 term, and, therefore, the matrix of quadratic form is

$$\mathbf{L} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}.$$

[2] This complete theorem can be found in [Kol84].

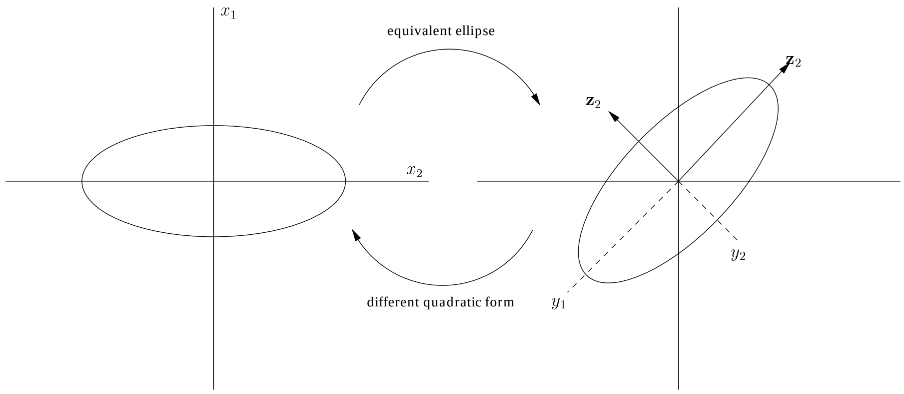


Figure 4.4: Similar quadratics

It follows that the matrix of the quadratic form will be diagonal if and only if its corresponding quadratic form has no term with respect to the non-principal axes [LE96]. So we seek to find $\mathbf{z}_1$ and $\mathbf{z}_2$, the defining vectors for the coordinate system y_1 and y_2, because our resulting quadratic form is simpler and describes our right-most ellipse in terms of the principal axis (uncorrelated) versus the non-principal axes in the coordinate system of x_1 and x_2 (correlated). Mathematically we seek

$$\mathbf{AZ} = \mathbf{LZ} \tag{4.2.1}$$

where

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_n \end{bmatrix}$$

As it turns out, this a classical eigenanalysis problem, where we are looking for a set of orthogonal vectors and a diagonal matrix that describes the original matrix in simpler terms. The columns of $\mathbf{Z}$, Equation (4.2.1), are called *eigenvectors* and the diagonal values of $\mathbf{L}$ are the *eigenvalues*, each eigenvector will correspond with its defining eigenvalue, i.e. $\mathbf{A}\mathbf{z}_i = \lambda_i \mathbf{z}_i$ [LE96].

Now recall that the eigenvalue matrix will explain the quadratic form with respect to the orthogonal eigenvectors. In terms of ellipsoids, this implies that large eigenvalues will correspond to the principal, or major, axes, and smaller eigenvalues will be the directions of minor axes. Because of this observation we will order our eigenvalues such that the largest eigenvalues are first or

$$\lambda_1 > \lambda_2 > \cdots > \lambda_n.$$

Thus the major axes of the ellipsoid occur first followed by minor axes. This ordering will allow us to make a relation between data variability and principal axes of variation in the next section. First, although, it is important to emphasize the significance of what just happen. That is we rotated our view of the non-standard form ellipse so that we could look at it in standard form. To do this we needed to find the principal axes of the ellipse, which are orthogonal. These properties will allow us to perform dimension reduction and derive a smaller, p-dimensional, data element that will approximate our original n-dimensional vector.

4.2.3 Dimension Reduction

Now it begs the question: what is the connection between ellipses, arbitrary datasets, and dimension reduction? Answering this question involves changing our data into a alternative representation, such that we can find the principal directions of the dataset. Expressed alternatively, we seek to find the principal vector that fits the data best, in a least squares sense. After finding this vector, we wish to find the next vector that best fits the data, yet is orthogonal to the first vector [Jac91]. This is repeated until no more orthogonal vectors can be found. Figure 4.5 clearly demonstrates this goal in two dimensions. The first vector, $\mathbf{z}_1$, is the vector of best fit through the dataset, and the second vector $\mathbf{z}_2$ is the next best fit to the data; yet, orthogonal to $\mathbf{z}_1$. Because there are no more vectors that fit — in the least squares sense — to the data and are orthogonal to the first two, we are done.

The relationship between this and the previous section is obvious. We only need to find a way to express our data in a quadratic form. Coincidently, the covariance matrix of arbitrary, n-dimensional data defines a hyperellipse in n-dimensional space [Gar99]. So, practically, we only need to form this covariance matrix and perform the eigenanalysis on that matrix to find the uncorrelated data elements. The corresponding eigenvectors will be the directions of variability because, in the least squares sense, the directions of variability will be the best fit vectors.

Composing the covariance matrix is a matter of finding the covariance between the i^{th} and j^{th} attribute values for the i^{th}, j^{th} entry in the matrix. Specifically, the

$$\mathbf{C}_{i,j} = Cov(\mathbf{d}_i, \mathbf{d}_j)$$

where

$$Cov(\mathbf{d}_i, \mathbf{d}_j) = E(\mathbf{d}_i\mathbf{d}_j) - E(\mathbf{d}_i)E(\mathbf{d}_j).$$

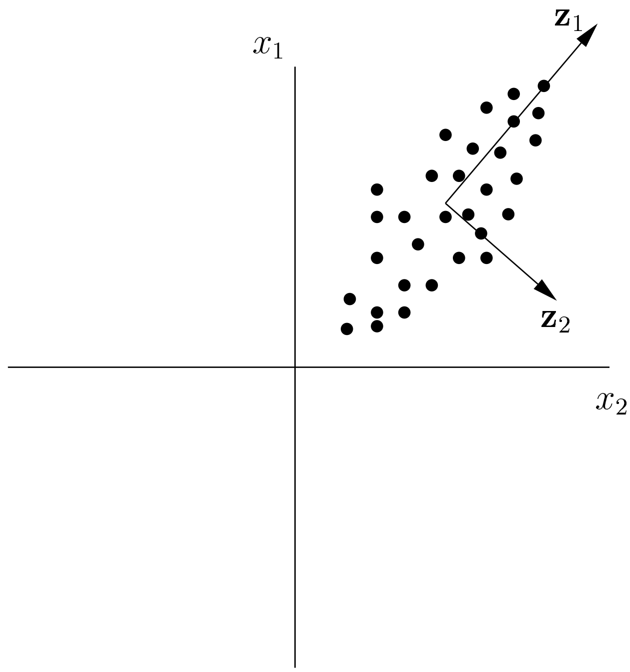


Figure 4.5: Fitting orthogonal vectors to arbitrary data.

And E is the expected value function.[3] However, the covariance matrix is not sufficient for an arbitrary dataset [Jac91] because a given dataset is not guaranteed to have common units among its variables. We need to either standardize our data prior to making the covariance matrix, i.e. initialize our data to fit within the unit cube in R^n, or compose the correlation matrix, where correlation among variables is analyzed, and data is standardized using standard deviations of the data. That is

$$\mathbf{Cor}_{i,j} = \frac{\mathbf{C}_{i,j}}{\sigma_i \sigma_j}$$

where σ_i is the standard deviation of attribute i.

Regardless of which representation we use, the concept remains; that is we perform the eigenanalysis on the given matrix to find the directions variability and "best" fit to the dataset. Once in this form, the eigenvectors, $\mathbf{Z}$, will allow us to perform our dimension reduction. That is, we can take a subset of the eigenvectors and project our data into a subspace of R^n. This property is a direct result from the eigenvectors being orthogonal, which, by properties of linear algebra, also makes the eigenvectors a basis for the vector

[3]Here we use subscripts as indices. The $\mathbf{d}_i$ is the i^{th} attribute for $\mathbf{d}$, $\forall \mathbf{d} \in D$.

space in R^n [LE96]. Basis vectors have this interesting property that they naturally form subspaces within their defined vector space. For illustrative purposes let $\mathbf{Z}_n = \mathbf{Z}$, where n number of column vectors in $\mathbf{Z}$. If we choose the set of column vectors from $\mathbf{Z}_n$ such that $\mathbf{z}_i^T \mathbf{z}_j = 0$ then these vectors form a vector space in $\mathbf{Z}_{n-1}$. We can repeat this process and find a subspace $\mathbf{Z}_{n-2}$ or more generally,

$$\mathbf{Z}_n \supset \mathbf{Z}_{n-1} \supset \cdots \supset \mathbf{Z}_1.$$

Our interests are in eigenvectors that point in the largest directions of variability. Because it is in this subspace that the majority of the data properties are described. That is we can throw away those directions that do not account for that much variability.

To determine how much variability is explained by a set of eigenvectors, we only need to look at the proportions of the eigenvalues. We let the user define a level of variability that the user wishes to retain from the original dataset, τ, such that,

$$\frac{\sum_{i=1}^{p} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \approx \tau. \tag{4.2.2}$$

We are left with a subspace $\mathbf{Z}_p \subset \mathbf{Z}_n$ that captures approximately τ percent of the variability. Jolliffe [Jol86] and Jackson [Jac91] describe other methods to pick the statistically best p. However, we wish to retain the flexibility that the *proportion of trace* gives us. In the end, we hope p is significantly smaller than n (suggesting that most of the variability can be captured in a few components).

This set of eigenvectors is formally known as the principal components, and once the principal components that account for τ are determined, we can generate the component scores for use in simplification, which are simply the projected data points. Generating the principal component scores is simply a matrix-vector multiplication on the original data element, $\mathbf{d}$. If the data is not in standard form we need to convert the data into standard form prior to applying the transformation (this can be done by scaling the data by the standard deviations). If the data has been normalized to the unit cube then we only need the eigenvector matrix. Regardless, we will generalize the matrix in both cases as simply $\mathbf{Z}$ with p eigenvectors. So the computed scores can be formed as follows:

$$\mathbf{y} \;= \mathbf{Z}^T \left(\mathbf{d} - \bar{\mathbf{d}}\right) \tag{4.2.3}$$

where $\bar{\mathbf{d}}$ is mean data element of our dataset. Conceptually, this mapping rotates our data element such that it is now relative to the principal component axes defined in $\mathbf{Z}^T$.

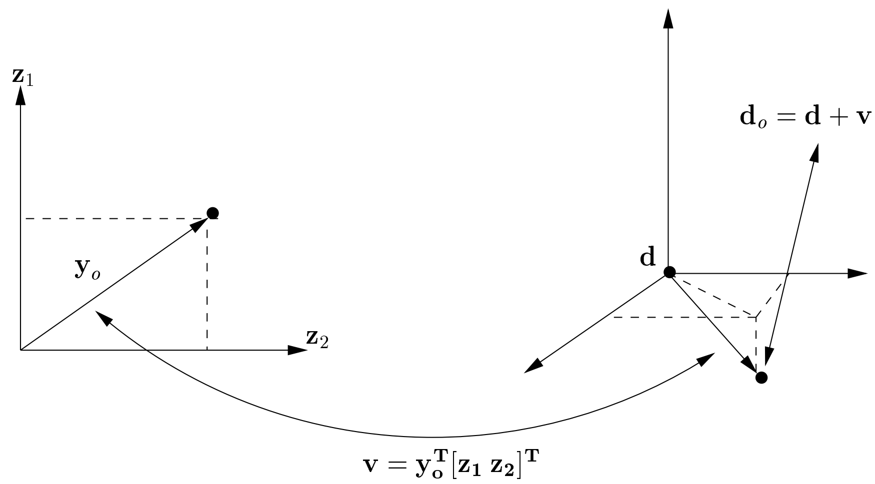


Figure 4.6: Unwinding PCA calculation

Thus, we have this nice simple transformation that projects our data into a lower dimensional space. Furthermore, the projected data points represent *most* of the variability present in the attribute space. However, having our vectors projected in this lower dimensional form presents a problem. Specifically, as we mentioned, some appearance preserving simplification algorithms find optimized data elements. Fortunately, as it turns out, we can manipulate (4.2.3) to form another equation that will map our data values in principal component space back into the original space.

4.2.4 Reversing Dimension Reduction

Before we begin the derivation of this equation, we would like to give a little intuition behind its inner workings. First, note that the matrix in (4.2.3) simply rotates and scales our data in some way. Thus a vector in principal component space has a relative length equal to the same vector in our original space. So if we have a vector in principal component space, we can map the movement in this space to a vector in our original space. Figure 4.6 illustrates that because we know how our components are weighted, we can form some, approximate, mapping back into our original space. In this example, $\mathbf{z}_1$ and $\mathbf{z}_2$ is the subspace that contains the projected data point, $\mathbf{y}_o$. The components of $\mathbf{y}_o$ indicate the direction and magnitude taken in this space. So the corresponding vector in the unprojected space, $\mathbf{v}$, is a linear combination of the components of $\mathbf{y}_o$. We only need to anchor the vector appropriately to find $\mathbf{d}_o$.

More generally, let $\mathbf{y}_o$ be our optimized vertex given to the principal component module, and let $\mathbf{y}$, from Equation (4.2.3), be a value that is close to a value in our original space. We seek $\mathbf{d}_o$; therefore, given Equation (4.2.3) we can compose the following:

$$\mathbf{y}_o - \mathbf{y} = \mathbf{Z}^T \left(\mathbf{d}_o - \bar{\mathbf{d}}\right) - \mathbf{Z}^T \left(\mathbf{d} - \bar{\mathbf{d}}\right) \tag{4.2.4}$$

Or,

$$\mathbf{y}_o - \mathbf{y} = \mathbf{Z}^T \left(\mathbf{d}_o - \mathbf{d}\right) \tag{4.2.5}$$

If we assume that $p = n$ then

$$\mathbf{Z}\left(\mathbf{y}_o - \mathbf{y}\right) = \mathbf{d}_o - \mathbf{d} \tag{4.2.6}$$

because our $\mathbf{Z}$ vectors our orthonormal. Thus we can unwind our values by using the following equation:

$$\mathbf{d}_o = \mathbf{Z}\left(\mathbf{y}_o - \mathbf{y}\right) + \mathbf{d} \tag{4.2.7}$$

If $p < n$ then the equality does not hold and our matrix multiplication in the solve step of (4.2.6) represents the amount of variability explained [Jac91] in each of the respective variables. Or conversely, the amount of variability unexplained. Therefore, if a user chooses a τ of 90% then they are willing to give up approximately 10% of the variability to examine the dataset.

Ideally there are some flaws with the above statement. The variability of some variables in the lower dimension may be completely explained by p principal components, e.g. 99%, where other variables may lack significant explanation from the p principal components, e.g. 50%. But those variables that are mostly explained are the most variant attributes, and, therefore, a large deviation in the least variant attributes does not affect the unwound vectors. If one wished to ensure a better representation of the dataset one should use a better stopping rule presented in [Jac91], usually this implies inclusion an additional component. For instance, some stopping rules are based on the amount of residual variability one is willing to accept.

4.3 Additional Remarks

We now have a suitable method to approximating our original n-dimensional data element in a much lower dimensional space. The user is given the flexibility to choose the amount of variability he wishes to retain, and p principal components are chosen with respect to this desired amount. In most cases, the highest amount of variability will be present in a only a few dimensions, and thus our error metric can be evaluated at a very low computational cost. Naturally, as a tradeoff, we lose some of the representation of the data space through this conversion. But we have limited this loss in only those directions of low variability. We also described a method that allowed a transformation between the two spaces. This is particularly important when the chosen mesh simplification algorithm returns an "optimized" vertex position, and it becomes necessary to unwind this point back into the original space.

In the upcoming chapter, we will iterate how to use this principal component module within a particular appearance preserving mesh simplification algorithm. We will also demonstrate our implementation of PCA within three popular algorithms. Results will show that, despite the loss in dimensionality, key features of the dataset are preserved.

Chapter 5

Implementation, Analysis, and Results

In the previous chapter, we introduced the notion of approximating a n-dimensional data elements with a p-dimensional counterpart. This was achieved by finding the principal components of the dataset, and projecting the elements onto the p-dimensional space. This chapter will show the results of our implementation of PCA using three popular mesh simplification techniques. The first two are based on the quadric error metric [Hop99, GH98] and the third uses the energy function described in [Hop96]. In each of these implementations, we will show how each algorithm performed with PCA versus without PCA. However, first we will overview the implementation and analysis of PCA in the above algorithms.

5.1 Implementation and Analysis

Mesh simplification is a two stage process. You have an initialization phase and a decimation phase, i.e. actual simplification. We add a third stage to this pipeline, a precomputation phase. We classify this precomputation phase as "offline" work. Work that can be performed once and is only necessary to be performed once. Arguably, some, if not all, of the initialization phase can be classified a precomputation work, but this is algorithm dependent, and because we seek a general classification of all algorithms, we must isolate these two steps.

With any algorithm, the first stage should be to compute the principal component matrix, if it has not already been precomputed. In our implementation, the principal components matrix is computed each time the program is executed. However, it is entirely possible to precompute the principal component matrix upon acquisition of the data, i.e. we can put the cost of computing the principal components as an "off-line" cost. In a sense, we can attach, to all datasets, a principal component matrix unique to that data. For the life of the dataset, this principal component matrix could exist with a small storage cost of $\theta(n^2)$ (considerably less than that total size of a typical dataset). Additionally, in terms of computational cost, this principal component matrix accounts for $O\left(n^2N + n^2\right)$ operations, where N is the number of data elements, and N will typically be large, at least larger than n. As N becomes much larger the cost becomes almost linear, but, again, this cost can be consider negligible because it can only occur once, off-line, for the entire life of the data.

The next step, with any algorithm, is select the components that account for the given τ. We can use Equation (4.2.2) to find this τ during the algorithm's initialization phase. It is important that this operation be performed first, in the initialization phase, because other operations may be dependent on the chosen scores.

In a quadric based error metric scheme, this additional front-end computation results in significant savings elsewhere. Specifically, to take advantage of dimension reduction, we substitute all instances of $\mathbf{d}$ with $\mathbf{y}$, as in (4.2.3). So $\mathbf{Q^d}$ becomes $\mathbf{Q^y}$. (Note that it is entirely possible to use the entire data element vector, attributes and geometry, in PCA calculation, but it is better to use PCA with respect to attribute information only, as was in our experiments.). Now, if y-score substitution is done on the fly then we add, in terms of computation cost, at most $O\left(Npn\right)$ operations to the initialization phase of the algorithm. This may seem considerable, but p will usually be very small. Moreover, in reduced dimensional space the cost of evaluating our quadrics become $O\left(p^2\right)$ versus $O\left(n^2\right)$, using Garland's metric, and $O\left(p\right)$ with Hoppe's.

Hoppe's quadric is quite good in a multidimensional environment, but PCA adds an additional level of flexibility not previously available. For instance, a dataset may be so large, N or n, such that loading it into memory would be impossible. In this instance, PCA could be useful when p is small, almost negligible, compared to n. Moreover, a p dimensional element can also reduce the total running time. Although, here we would not want to perform our y-score substitution on the fly because we would be incurring more cost than the algorithm originally imposed. Instead, before the data is brought into the simplification algorithm we could convert the data into the y-scores. After simplification, we merge the

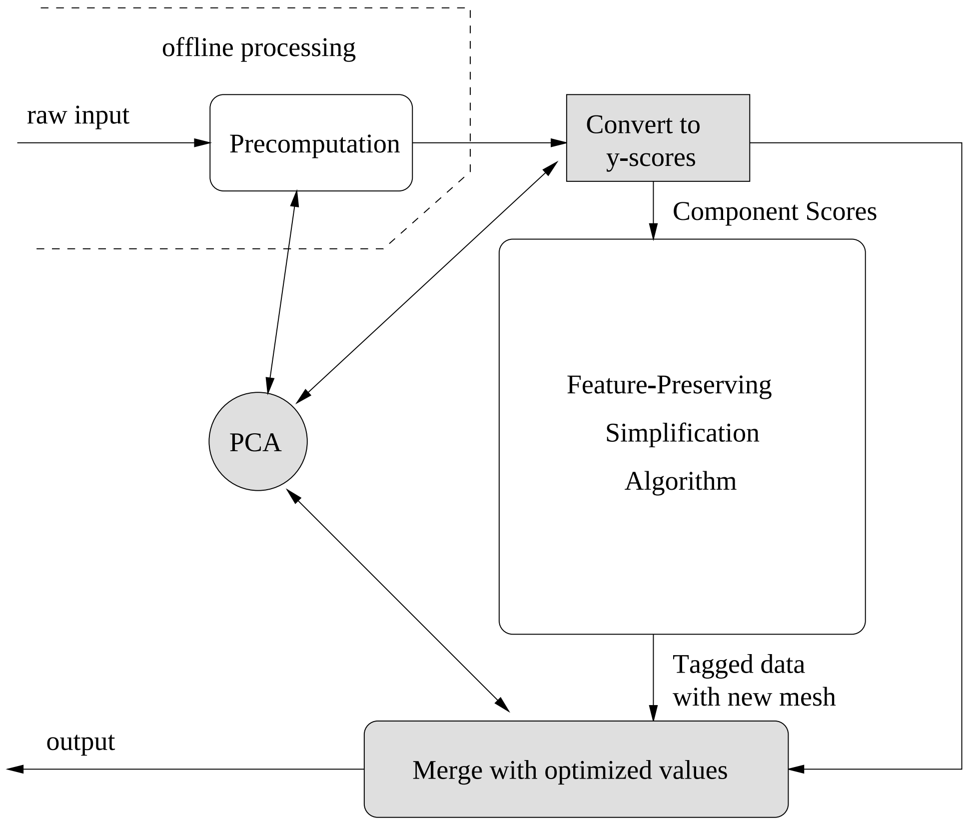


Figure 5.1: A possible tagging implementation.

simplified data with the old data back into its original form. We can do this by simply *tagging* elements that are suppose to be removed but not actually removed, see Figure 5.1. When simplification finishes, the algorithm can merge those non-tagged elements with the original dataset. The intersection between these two sets will be the simplified dataset. Additionally, we can associate a special tag with those elements that were "moved" during simplification. These elements are special because they must go through an unproject stage, where we have to apply Equation (4.2.7) to find the unprojected point in the original space.

To further demonstrate the flexibility our extension, we provide implementation details for a third metric based on Hoppe's energy function [Hop96]. Recall that this method measures the energy from the old mesh to the new mesh. The energy itself is calculated by taking the original set of points and measuring the distance from these original points to their projection onto the simplified mesh. Energy of attribute space is based on the attribute values of this point of projection onto the mesh. Using principal components analysis in this method is straight forward. Simply substitute the attribute values for the original points, i.e. the points to be projected onto the new mesh, with their principal component counter parts. We can do this because these points are only used to calculate error, and, therefore, do not need to remain in their original space. During simplification we can again substitute on the fly when necessary. Additionally, a tagging implementation, like above, can also be used.

It is important to emphasize that regardless of the type of algorithm you are using, the use of PCA can improve performance, when dealing with high dimensionality. PCA only requires an additional precomputation step, and the results of that precomputation can be stored for the life of the dataset, giving us a compact representation of the whole dataset. In our implementation we created a principal component module. A simplification algorithm can query this module for data elements. If PCA mode is turned on, i.e. the mode that invokes data conversion, then the module returns the converted element.

Integrating locks in a simplification algorithm is also simple. Prior to evaluating an edge or a data element, we can check a bit flag. If this flag is turned on then the element is locked. We can simply ignore the data element, or, in the case of edge collapses, force the collapse onto the other element.

5.2 Results

This section serves as a "proof of concept" of our work. We implemented the three algorithms discussed above and tested those algorithms on an arbitrary dataset with and without principal components analysis. We also performed numerical and visual tests to verify that our component was working. In this section we will illustrate the tests performed and provide visual results to justify our work.

The dataset was obtained from the Climate Research Unit at the University of East Anglia [Cli00]. The data consists of half degree mean world climatologies for the month of January from 1961-1990. At each half degree increment there are eleven attributes:

a. mean temperature — average daily temperature.

b. precipitation — monthly precipitation.

c. vapor pressure — average pressure.

d. wind speed — maximum daily wind speed averaged over the month.

e. frost frequency — the number of days ground was covered with frost.

f. diurnal temperature range — temperature difference between night and day.

g. cloud cover — percentage of the sky covered by clouds.

h. maximum temperature — average maximum temperature.

i. minimum temperature — average minimum temperature.

j. radiation — the amount of solar radiation received.

k. wet day frequency — the number of days with precipitation.

Now remember that these attributes are the averages for the entire month of January. For example, mean temperature is the average daily temperature averaged over 29 years. For illustrative purposes, we extrapolated a section of North America, (76n, 166w) to (8n, 76w). In total, the dataset consisted of just over $10,000$ data elements with about $20,000$ faces (after triangulation). This particular dataset was choosen because a novice can quickly identify with weather data.

Before we begin, it is interesting to analyze the attributes included in the above dataset. That is, some of the attributes are closely related to each other. For example, one would guess that the temperature components would be highly correlated. Therefore, within these components, there is a lot of redundancy encoded, and a method like principal components analysis should rid us of this extra data. In fact, most weather data will fall into this category. Specifically, there will be a number of attributes that are highly correlated, with low covariance, and it is by "throwing-out" those variables we can gain a suitable approximation.

Figure 5.2 illustrates the above property. This figure illustrates the color maps of each of the above attributes; each attribute label matches the above enumerated list. Also included in the figure are the first four principal components scores. Before score computation, each attribute was scaled to the range $[0, 1]$, to invoke scale-invariance. The color maps are perceptually balanced with darker, blue/green, colors representing the low values, i.e. close to zero, and the brighter, red/pink, colors represent values at the upper spectrum, i.e. values close to one. Likewise, the principal component scores have normalized values between

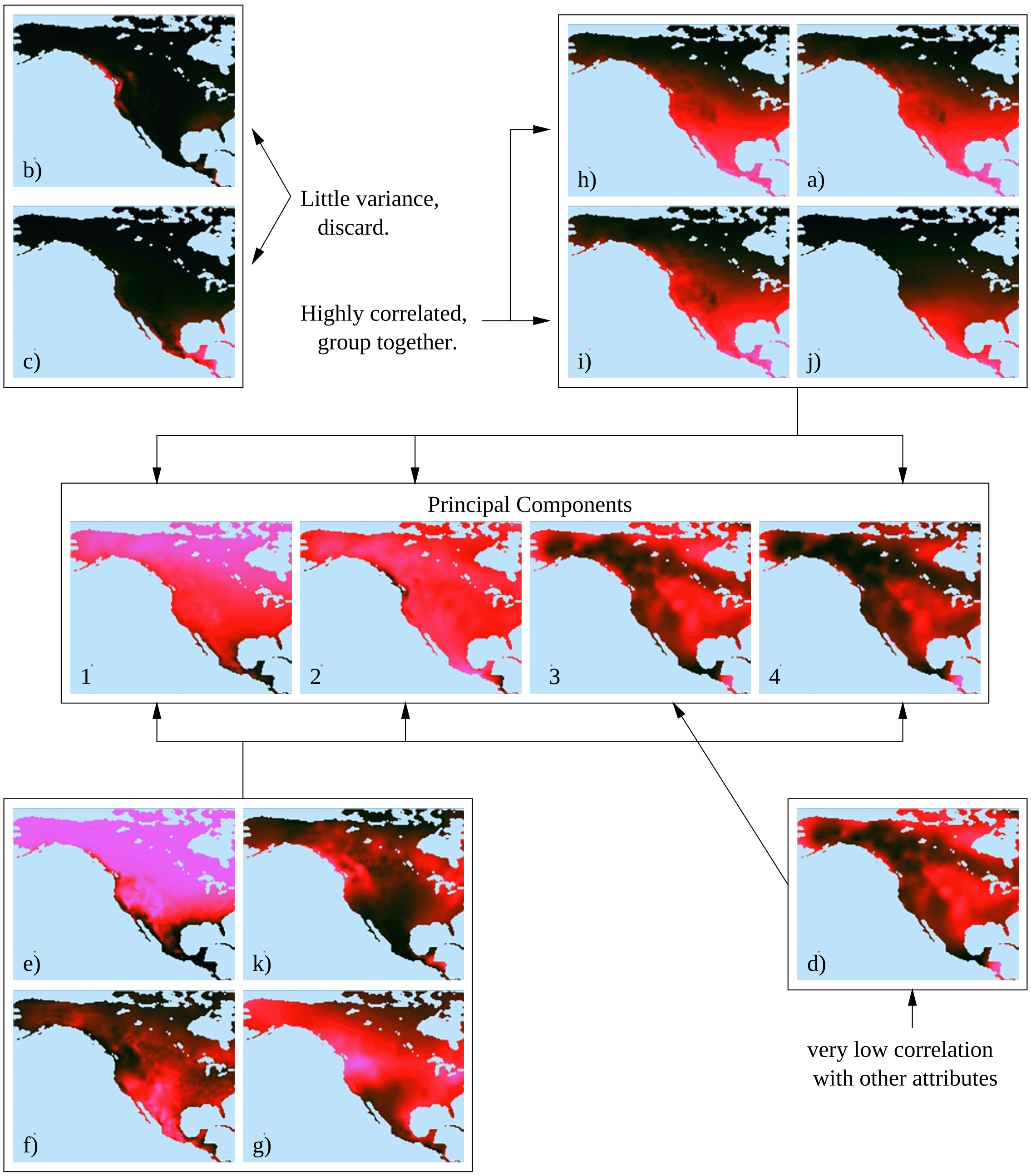


Figure 5.2: Principal components of the CRU dataset.

zero and one with balanced color maps. By examining the respective attribute values one can notices interesting properties of the dataset. For instance, there are a few attributes, namely precipitation and vapor pressure, that hardly vary. Consequently these directions of variability will not carry much weight and, therefore, can be excluded in the computation of error. Other attributes, like the temperature components, are highly correlated and can be encoded together. And there may exist a few, one in this case, that has no, or has very little, correlation with the other attributes, i.e. the variance in this dimension follows a principal axis. The other attributes are correlated in some way, but it is not entirely obvious from the color maps. Regardless, some combination of these components will define the vectors that make up the principal components. We use arrows to indicate that some linear combination of these attributes will give us the resulting scores. Keep in mind it may be the case that only a small percentage of the attributes factor into the scores, but still there is some present. The remaining, least variant parts would participate in the remaining scores, not shown.[1]

The above principal component scores account for approximately 93% of the variability of the dataset. Component wise, this percentage breaks down into the following percentages: $\lambda_1 = 54.4\%$, $\lambda_2 = 20.9\%$, $\lambda_3 = 10.5\%$, and $\lambda_4 = 6.8\%$. Our user defined target tau was 90%. So it took four principal components to meet or exceed that value. Notice that we could of excluded the last component and still nearly meet our desired tau. The inclusion of the last component, which typically will have considerably less represented variance than the previous components, will depend on how close to the desired tau one wishes.

These four components were used for the remaining test cases. Notice that the dimensionality of our approximated data space is considerably lower than the original eleven attribute we started with, about a third of the size. And we suspect that this change in dimension will allow us to save time during simplification without sacrificing much in terms of additional error during simplification.

5.2.1 Execution Timings

This section will demonstrate the relative time performance gain using the computed principal components vis-á-vis the original space. To compute the timings, we ran our extensions with each of the implemented algorithms, i.e. we simplified using the principal component

[1] For illustrative purposes only, we have included the correlation matrix of these components in Appendix A. Each entry corresponds to the correlation between the i^{th} and j^{th} attribute.

Time(sec)				
	O_GQEM		GQEM	
% Simp	w/o PCA	w/ PCA	w/o PCA	w/ PCA
90	67.912	13.470	7.665	4.138
80	57.364	11.225	6.530	3.566
70	48.772	9.550	5.583	3.401
60	42.509	8.237	4.726	2.571
50	33.895	6.649	3.911	2.122
40	27.000	5.362	3.105	1.680
30	20.291	3.933	2.310	1.243
20	13.357	2.583	1.529	0.831
10	6.656	1.306	0.776	0.414

Table 5.1: Timings for GQEM with PCA.

scores as above. We then ran those same algorithms with the full dataset. Additionally, each test was run on various levels of simplification to see the effects of more simplification versus time and error. For our illustrations we will use the following abbreviations: Progressive Mesh Energy Function (PM), Hoppe's Quadric Error Metric (HQEM), Garland's Quadric Error Metric (GQEM), Garland's Quadric Error Metric with optimal placement policy (O_GQEM). Additionally we will tag the abbreviation PCA for those tests that used principal component scores.

Now the below tables, 5.1 and 5.2, are the results of our timing benchmarks. Each number is the total time of simplification in seconds. The leftmost column indicates the percentage of simplification performed, i.e. 90% implies only 10% of the data remains. Table 5.1 shows the benchmarks for Garland's quadric error metric using both endpoint/midpoint and optimal placement policy. As you can notice the optimal placement calculation carries a high computational cost that will continue to impact simplification performance as the number data elements, and attributes increases. By compressing this space down we gain a significant improvement in speed. Moreover, even using just endpoint placement we notice an improvement in performance. Likewise, in Table 5.2 we can see comparable time savings. The nice property about using PCA is that as the number of attributes increase the number of principal components increases at a much slower rate. Of course this is assuming you have some correlation in the dataset.

Time(sec)				
	HQEM		PM	
% Simp	w/o PCA	w/ PCA	w/o PCA	w/ PCA
90	6.387	4.230	660.779	592.245
80	5.208	3.560	424.773	398.380
70	4.417	2.972	326.127	309.417
60	3.670	2.502	259.855	243.109
50	2.993	2.25	197.676	194.467
40	2.374	1.606	142.655	147.473
30	1.778	1.199	102.62	104.438
20	1.184	0.797	65.91	65.999
10	0.598	0.398	31.801	27.557

Table 5.2: Timings for HQEM and PM with PCA.

5.2.2 Quantitative Results

Given the savings in execution time, we were interested in how much loss, in terms of error, occurred by using the approximated data element versus the full attribute space. In other words, will the loss in dimensionality do more damage than good. Mesh simplification is the process of removing as many data elements as possible inducing the least amount of error. So, as expected, there is some level of error present from our original, fully populated, mesh to our new, less populated, mesh. By applying our new extension we expect some new error introduced by simplification, but we do not want this new error to far exceed the advantages in time.

There are many common approaches to measure the error between two meshes. The most common approach taken is the Hausdorff distance [GH98]. That is, we wish to find the distance between the new mesh and the original mesh. To find this “distance” we must take all of the points in the original mesh and find the average distance to the new mesh. Moreover, because this operation may not be symmetric we need to apply the reverse as well. Mathematically, we can say that the Hausdorff distance between two meshes is

$$\frac{1}{N_a + N_b}\left(d(M_a, M_b) + d(M_b, M_a)\right) \tag{5.2.1}$$

where

$$d(M_a, M_b) = \sum_{\mathbf{x}_p \in M_a} min||\mathbf{x}_p - \mathbf{y}_p||$$

such that $\mathbf{y}_p \in M_b$. Here N_a and N_b are the number of elements in mesh M_a and mesh M_b, respectively, and $\mathbf{x}_p$ and $\mathbf{y}_p$ are the positional parts of the data elements, $\mathbf{x}$ and $\mathbf{y}$. To find the error within the attribute portion of the data element, we measure the difference, like Hoppe [Hop99], between the interpolated attribute values at $\mathbf{x}$ and $\mathbf{y}$. To quantify attribute, represented by a subscript a, error at these points we apply three measurements:

$$L_1 = \sum_i |\mathbf{x}_{a_i} - \mathbf{y}_{a_i}|, \tag{5.2.2}$$

$$L_\infty = max|\mathbf{x}_{a_i} - \mathbf{y}_{a_i}|,$$

and

$$L_2 = ||\mathbf{x}_a - \mathbf{y}_a||$$

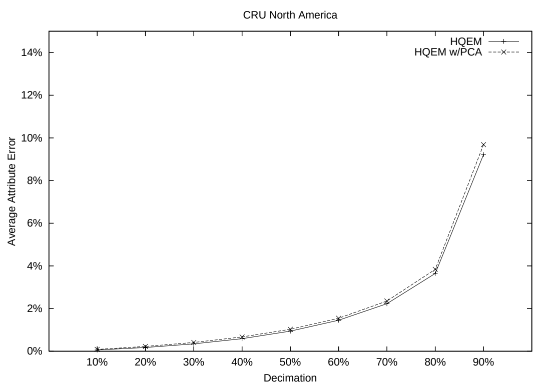
norms. For our results we found the maximum and average norm in each case, e.g. the maximum L_∞ norm is the maximum error in a single attribute over the entire dataset, and the average L_∞ norm is the sum of maximum error per data element divided by the number of samples taken. Averaging these values gives us a similar equation as in (5.2.1). Additionally because the distance is related to the distance squared we simply measured distance squared error versus the distance. Note that this still is a norm in R, therefore, is a valid metric. However, to differentiate, we will use the notation L_2^2 to represent this norm.

Figure 5.3 shows the averaged L_1 norms for the CRU dataset. This norms have been scaled to represent the percentage of error present.[2] Each of the four graphs follow directly from the timing results taken above. As it is clearly evident, despite the reduction in dimensionality the average error does not deviate far from the error initially imposed by the algorithm. In fact the differences in error is barely noticeable. The difference is only really noticeable as the percentage of simplification increases. In some cases, the PCA version performed better at the higher end of simplification, see Figure 5.3(d). This was likely caused by some degeneracies along the edges of the mesh that did not appear in the PCA version.[3]

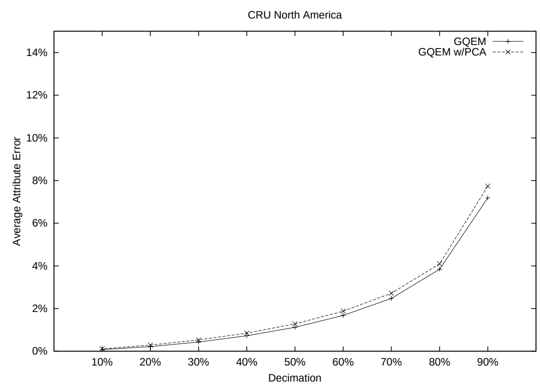
Naturally, as more principal components are included the closer this error curve will approach its non-PCA counterpart. Likewise, the fewer principal components will imply a larger gap in-between these two curves. Regardless, we are receiving significant savings in the execution time versus the relative loss in error.

[2] We can easily scale these values because we are working within the unit cube in R^n.

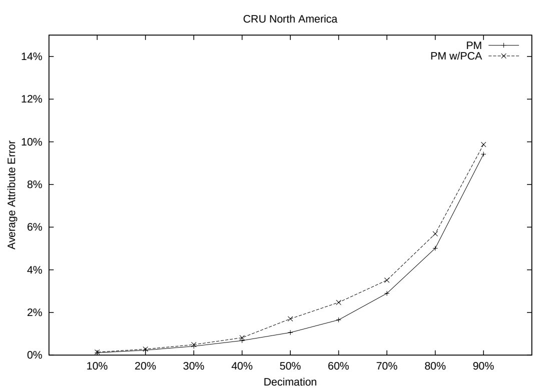
[3] A degenerate mesh has distorted boundaries.



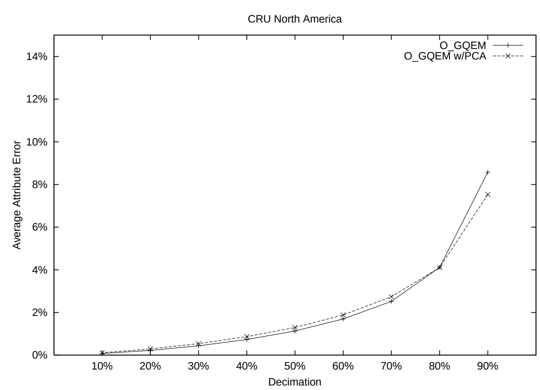
(a)



(b)



(c)



(d)

Figure 5.3: L_1 norm from original mesh to the version.

Meth	Geom L_2^2 (10^{-4})	Attr Avg L_2^2 (10^{-2})	Attr Max L_∞
HQEM w/PCA	0.000129	0.00899	0.091480
HQEM	0.000128	0.00666	0.053568
GQEM w/PCA	0.000151	0.01691	0.195856
GQEM	0.000141	0.01219	0.120494
O_GQEM w/PCA	0.000150	0.01739	0.252785
O_GQEM	0.000138	0.01245	0.195856
PM w/PCA	0.083480	0.07752	0.390071
PM	0.056038	0.01703	0.390071

Table 5.3: Error computations for 50% simplification.

To further emphasize these results we have included the other norms for the 50 percent simplification case, see Table 5.3.[4] Interestingly, the maximum error for the PM case remained the same using the principal component scores. This implies that the principal component extensions are doing no worse than the non PCA counterparts, i.e. in the worst possible case. Additionally, the average difference in error over all norms taken, over all tests performed, is just under 1% at 0.89%. Indicating that the improvement in performance is not outweighed by the loss in error.

Aside, we had initially hoped that by using PCA would could get around some of the degenerate issues of a particular algorithm. Specifically, Hoppe [Hop99] states that Garland's QEM can actually underestimate the error. He states "[t]he effect is that **v** is generally not compared to the geometrically closest point, but some geometrically farther point that has a closer attribute value" (where **v** is the data element). We suspected that maybe a reduction in dimensionality would reduce this effect, but this may not necessarily be true. For example, let us assume that we have some attributes in our original space that don't vary at all, i.e. $var = 0$. The hyperplanes that are contained in this space all fall the same way in a particular dimension. So the effect of Hoppe's statement is only caused by other dimensions. Now if we vary these same dimensions slightly then the same net effect will hold, it is just that these other dimensions will have a *little* influence now, and the net effect of removing the least variant parts is small. This does not forgo the fact that we may gain some improvement in error computation, it is just that we should not always expect it.

[4] The full tables can be found in Appendix A.

5.2.3 Qualitative Results

To complement the quantitative results above, this section illustrates the qualitative benefits of our extensions. Specifically, we will show that even after a large reduction in simplification we still have visually preserved the underlying mesh. Additionally, we will "visualize" using a multivariate visualization technique and show how locking data elements can be used to eliminate planing, as well as preserving important data elements.

Figure 5.4 shows the original mesh and the simplified mesh using our extension. In this example, PM was used with a simplification target of 80 percent. The two top images represent the underlying mesh, and the bottom images are the corresponding color maps of temperature, precipitation, and frost. Note how the the original mesh is densely populated and obviously contains too much data. The second mesh, on the other hand, has fewer elements but still preserves the mesh. Also, you can notice the larger areas of simplification are in the regions of low our no variability. Areas, especially around Seattle, have been preserved because of the large variation change in that region. Also, areas in Mexico are preserved due to the high variation in within the frost frequency component, which is due, in part, to the mountain ranges in that area. Likewise, we see mesh elements "dense-up" around the Rocky Mountain range.

In retrospective, we hinted that precipitation had a very low total variation. So how did the algorithm know to preserve the high variation change? Actually, the variation boundary was accounted for in another component; thus, preserving that feature.

Now color meshes are convenient when we are interested in the underlying mesh, but our interests are in multivariate visualization. That is, our visualization system is based on the work by Healey [HE98]. Our pexels simulate painted strokes like that on a canvas. In fact, the system we used for experimentation is a nonphotorealistic visualization system (NPVIZ). In this system, brush strokes can vary in size, regularity, orientation, color, and density. To visualize multiple variables at one instance, we made the following visual mappings:

- *temperature* → *color* — bluer, darker colors imply colder temperatures, and red, pinkish colors are warmer regions.
- *precipitation* → *orientation* — horizontally positioned pexels imply no, or little precipitation, and vertical pexels are regions of high precipitation.
- *pressure* → *density* — less dense pexels contain regions of low pressure, and more

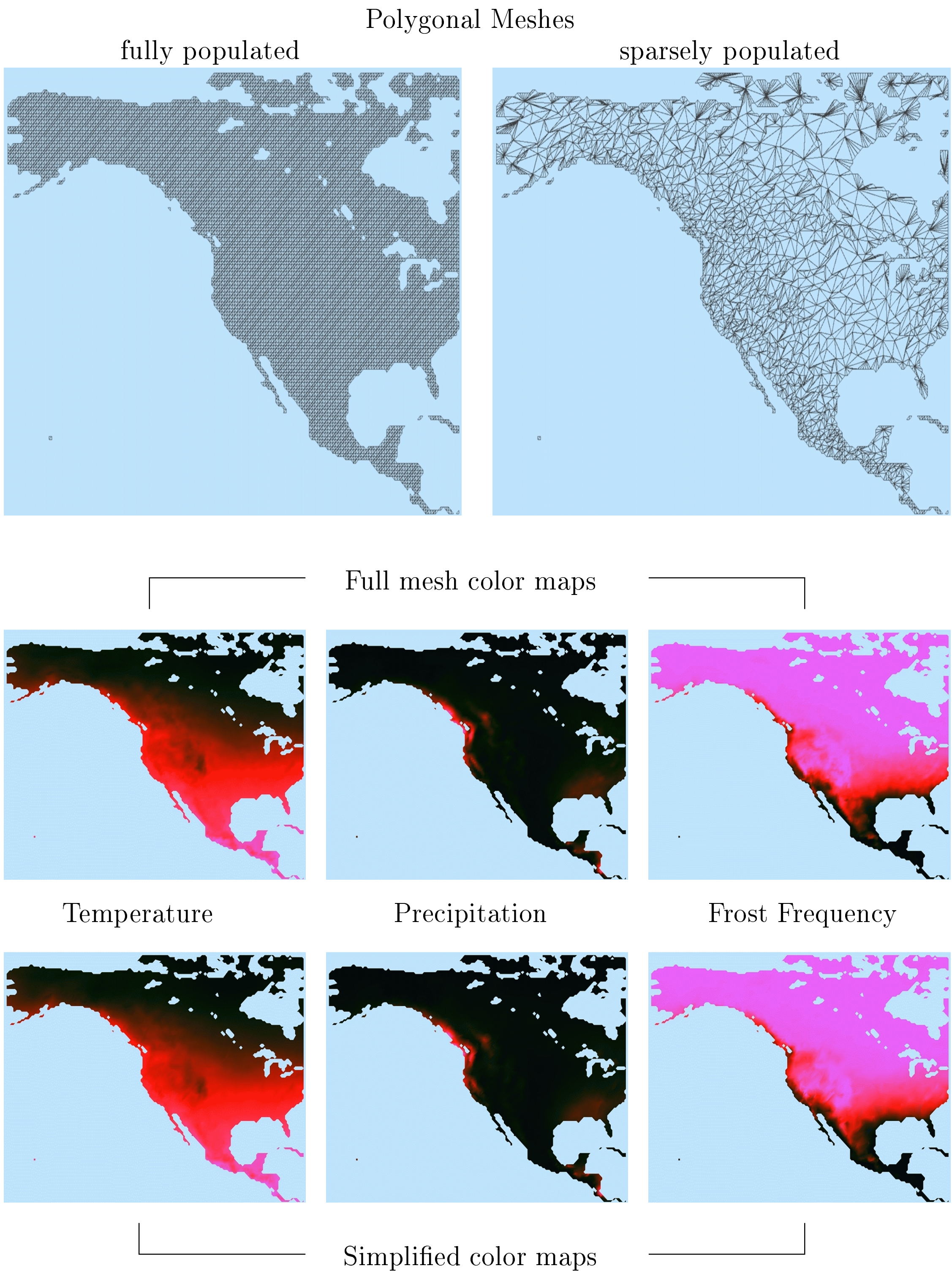


Figure 5.4: Simplified mesh with color maps

dense pexels imply high pressure values.

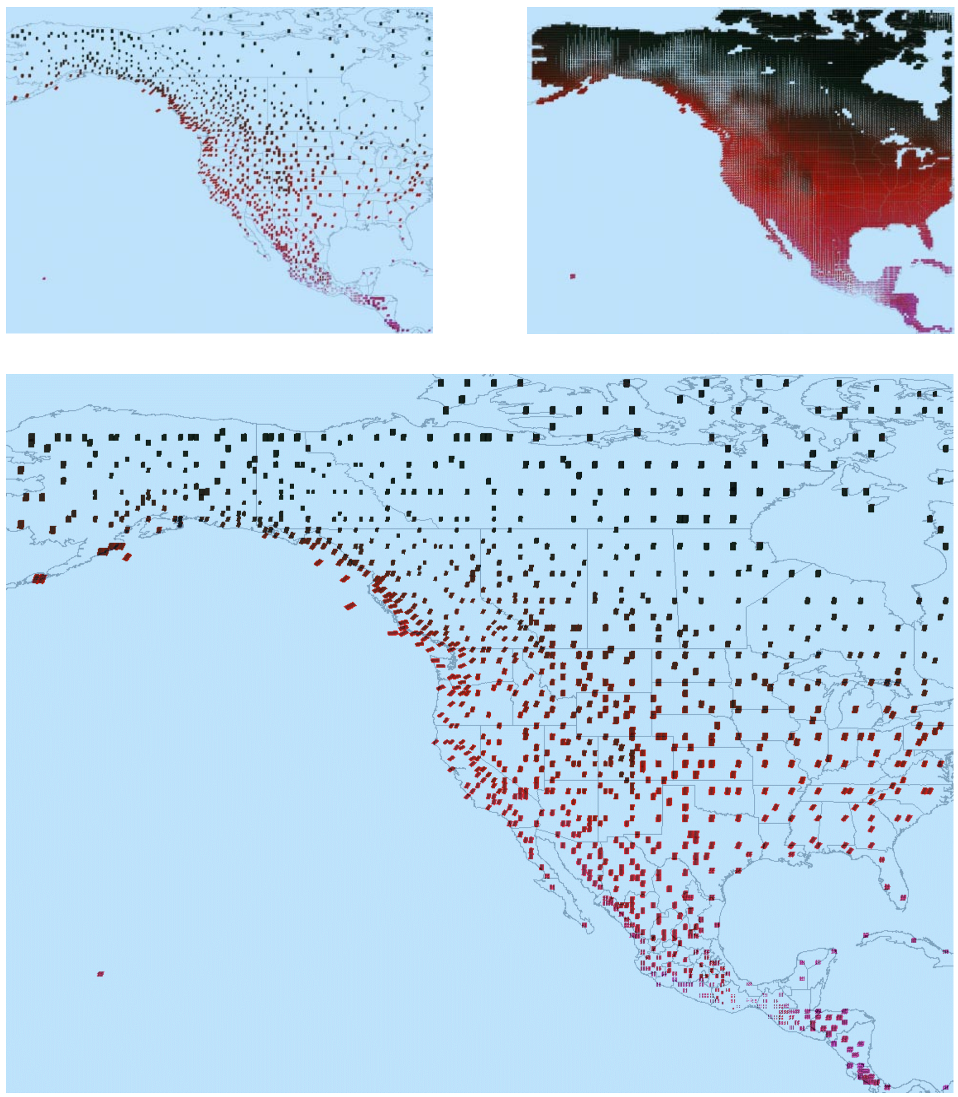
- *wind speed* → *size* — larger pexels are regions of low wind speed, and smaller pexels imply high wind speed.

We excluded regularity because it depends on a irregular placement of strokes over a regular field, and by performing simplification we invoke some irregularity in places that should not be irregular. Moreover, some would argue that because we are limited to visualizing a few attributes that we should just forgo simplifying all attributes simultaneously and only simplify those attributes that we are concerned with. But we would argue that large datasets prevent interactive visualizations. For example, a dataset with a large number of data elements would take considerably longer than the small dataset we have chosen for our examples. If the user was interested in multiple relations among attributes he would have to wait an equal amount of time per a change in mappings, whereas our method could simplify the entire dataset once.

Figure 5.5 is our visualization using the above mappings with NPVIZ. These simplified images were acquired by applying an 80% simplification using HQEM. The top right figure is the full dataset. Notice how this dataset is heavily populated and contains too much information. The left neighbor is the simplified version without using locking. You can notice features that are still emergent in this field. Specifically, you can see that mountain regions were preserved over the mid-western plains. Also, you can see that a majority of simplification occurred in the upper east portion of Canada. This is due to the low, smoothly varying data in those regions. However, there is a problem with this visualization. The user sees a huge amount of simplification, where it would be nice to have some indication of what is happening in those regions of large simplification. To eliminate this planing, we lock every fifth data element. This creates a more visually appealing effect, bottom image.

Notice how the areas that suffered huge amount of simplification still have data intact. Now the user can more readily see values in those regions. In a sense, locking has allowed us to achieve balance between simplification and the final visualization. This is especially true in point based visualization system because these systems do not have an underlying mesh.

No locking Full pexel array



Locking

Figure 5.5: Simplification locking

Chapter 6

Conclusion and Future Work

Large data will always be a problem in scientific visualization. However, we can take proactive approaches to curb the size of these enormous datasets. Our attempt to curb dataset size drew from previous work in computer graphics. We were able to take existing work within mesh simplification and apply it to a data simplification environment. We overcame the potential problems of using mesh simplification in this environment. Specifically,

- *Dimensionality* — because most algorithms were only designed for a small attribute space, we introduced principal components analysis to capture most of the variability in a much smaller dimensional space.
- *Significance of Spatial Location* — exact spatial location is not important in mesh simplification, so we introduced the notion of locks to "fix" data elements to their initial position. Also, this has the added benefit of eliminating planing.

In addition, our extensions are

- *General* — Principal components analysis and locking can be applied to any existing and future algorithms.
- *Computationally Sound* — Principal components analysis adds a minimal overhead cost that is only necessary to compute once.
- *Space Efficient* — We add very little cost storage overhead to the simplification process.

We should also add that our method is simple and flexible. That is, the principal component module can be completely isolated from the simplification algorithm. This particular strategy is advantageous because users can take working implementations and add a principal component "hook." For example a tagging based implementation would require very little overhead in terms implementation cost. The only requirement of this type of implementation is that the output data can be identified as removed, existent, or moved (moved vertices would have to be unprojected to find their landing point in the original space). Additionally, the principal component module is flexible enough to work on the fly. Data conversion could be requested whenever a data element is examined. Aside from dimension reduction, locking alone can be beneficial to simplification in a visualization environment, i.e. the user may want to preserve a specific set of points from being removed, and locking can be very simply integrated into a mesh simplification algorithm — with only the added cost of one bit of storage.

Results showed that we saved processing time and lost very little, in terms of error. But, without hesitation, there are still many avenues for future work. Some of the potential, future, avenues include:

- *Memoryless extensions* — Lindstrom and Turk [LT98] introduce the notice of recomputing per vertex quadric each time they are encountered, rather than carrying them through an addition operation. Their work shows improved simplification, in terms of error. A memoryless scheme is particularly important in data simplification environment, precisely because of the potential size of these sets.

- *Out-of-core extensions* — as we reviewed in previous chapters data can become large in both dimensionality and the number of data elements. It would be useful to create general extensions that would allow the use of current simplification algorithms with the large data, i.e. most current approaches work in primary memory. More recent approaches like [Lin00] perform out-of-core simplification, but more general approaches are needed that carry into all feature preserving mesh simplification algorithms. We would like to build a simplification engine that supports disk based simplification using a generic hierarchical data structure that can also plug-in to the various simplification algorithms.

- *Point-based simplification* — many datasets are not attached to a mesh; therefore, rather than converting the point based dataset into a mesh, perform simplification on the points. As the solution is not trivial, multiple regression could be used to analyze surface fit.

- *Addition of progressive mesh format* — In Hoppe's framework [Hop96], he simplifies the data into varying levels of simplification. Each edge collapse is recorded and can be "undone" to create a model that contains all levels of simplification simultaneously (in effect this mimics a multiresolutional model).

Certainly, there are other avenues of future work, but these are the most pressing for our interests.

Data acquisition methods will continue to improve, and our ability to visualize data will depend on our ability to parse this large body of information quickly. To allow the user to rapidly, effectively, and efficiently visualize their data, appearance preserving data simplification offers a plausible solution that allows a user to deal with large data so that a user can truly "see" all of their data.

Bibliography

[ACSW96] Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff, *Tioga-2: A direct manipulation database visualization environment*, 12th International Conference on Data Engineering, February 1996, pp. 208–17.

[Bau75] Bruce G. Baumgart, *A polyhedron representation for computer vision*, National Computer Conference '75, 1975, pp. 589–596.

[BCE+92] K. Brodlie, L. Carpenter, R. Earnshaw, J. Gallop, R. Hubbold, A. Mumford, C. Osland, and P.Quarendon (eds.), *Scientific Visualization: Techniques and Applications*, Springer-Verlag, Berlin, 1992.

[BGK+88] M. D. Brown, D. Greenberg, M. Keeler, A. R. Smith, and L. Yaegar, *The visualization roundtable*, Computers in Physics 2, 1988, pp. 16–26.

[BS96] C. Bajaj and D. Schikore, *Error-bounded reduction of triangle meshes with multivariate data*, SPIE vol 2656, 1996, pp. 34–35.

[CE97] M. Cox and D. Ellsworth, *Managing Big Data for Scientific Visualization*, ACM SIGGRAPH '97 Course 4, August 1997.

[CL93] Brian Cabral and Casey Leedom, *Image vectors fields using line integral convolution*, SIGGRAPH '93, 1993, pp. 263–270.

[Cli00] *Climate Research Unit: 1960-1990 Mean Climatologies*, http://www.cru.uea.ac.uk/cru/data/, December 2000.

[CLR90] Thomas Cormen, Charles Leiserson, and Ronald Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.

[CMO97] J. Cohen, D. Manocha, and M. Olano, *Simplifying polygonal models using successive mappings*, IEEE Visualization '97, 1997, pp. 395–402,564.

[COM98] J. Cohen, M. Olano, and D. Manocha, *Appearance-Preserving Simplification*, SIGGRAPH '98, 1998, pp. 115–122.

[CPD+96] A. Certain, J. Popović, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle, *Interactive multiresolution surface viewing*, Proceedings of SIGGRAPH 96 (August 1996), 91–98, ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[CVM+96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, *Simplification envelopes*, SIGGRAPH '96, 1996, pp. 119–128.

[dBvKOS91] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer, Berlin, Gr, 1991.

[Dix99] Alan Dix, *Human-Computer Interaction, 2nd ed.*, Prentice-Hall, 1999.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, *Multiresolution analysis of arbitrary meshes*, SIGGRAPH '95, 1995, pp. 173–182.

[EKSX96] M. Ester, H. Kriegel, J. Sander, and X. Xu, *A density-based algorthm for discovering clusters in large spatial databases with noise*, 2nd Int. Conference on Knowledge Discovery and Data Mining, 1996, pp. 226–231.

[EKX95] M. Ester, H. Kriegel, and X. Xu, *Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification*, 4th Int. Symposium on Large Spatial Databases, 1995, pp. 67–82.

[EM99] K. Erikson and D. Manocha, *GAPS: General and Automatic Polygonal Simplification*, ACM Symposium on Interactive 3D Graphics, 1999, pp. 79–88.

[FPSM91] W. J. Frawley, G. Piatetksy-Shapiro, and C. J. Matheus, *Knowledge Discovery in Database: An Overview*, AAAI Press/MIT Press, Menlo Park, CA, 1991.

[FR88] Bernhard Flury and Hans Riedwyl, *Multivariate Statistics: A Practical Approach*, New York: Chapman and Hall, 1988.

[FvDFH96] James D. Foley, Andreas van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, MA, 1996.

[Gar99] Michael Garland, *Quadric-based polygon surface simplification*, Ph.D. thesis, Carnegie Mellon University, 1999.

[GH97] M. Garland and P. Heckbert, *Surface Simplification Using Quadric Error Metrics*, SIGGRAPH '97, 1997.

[GH98] M. Garland and P. Heckbert, *Simplfiying Surfaces with Color and Texture using Quadric Error Metrics*, IEEE Visualization 98, 1998.

[GM92] A. Guide and W. Mass, *Advanced Visual Systems Inc*, 1992.

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Mesh optimization*, SIGGRAPH'93, 1993, pp. 19–25.

[HE98] Christopher G. Healey and J. T. Enns, *Building Perceptual Textures to Visualize Multidimensional Datasets*, IEEE Visualization '98, 1998, pp. 111–118.

[Hea92] Christopher G. Healey, *Visualization of multivariate data using preattentive processing*, Master's thesis, University of British Columbia, 1992.

[Hea96] Christopher G. Healey, *Effective visualization of large, multidimensional datasets*, Ph.D. thesis, University of British Columbia, 1996.

[Hea98] Christopher G. Healey, *On the use of perceptual cues and data mining for effective visualization of scientific datasets*, Graphics Interface 98, 1998, pp. 177–184.

[Hea99] Christopher G. Healey, *Perceptual techniques for scientific visualization*, SIGGRAPH 99 Course 6: Fundamental Issues of Visual Perception for Effective Image Generation, 1999.

[HG97] Paul S. Heckbert and Michael Garland, *Survey of polygonal surface simplification algorithms*, Tech. Report draft, Carnegie Mellon University, 1997.

[HM00] H. Hoppe and S. Marschner, *Efficient efficient minimization of new quadric metric for simplifying meshes with appearance attributes.*, Tech. Report MSR-TR-2000-64, Microsoft Research, June 2000.

[Hop96] H. Hoppe, *Progressive Meshes*, SIGGRAPH '96, 1996, pp. 99–108.

[Hop97] Hugues Hoppe, *View-dependent refinement of progressive meshes*, SIGGRAPH '97, 1997, pp. 189–198.

[Hop99] H. Hoppe, *New Quadric Metric for Simplifying Meshes with Appearance Attributes*, IEEE Visualization 1999, October 1999, pp. 59–66.

[HPvW94] Lambertus Hesselink, Frits Post, and Jarke van Wijk, *Research issues in vector and tensor field visualization*, IEEE Computer Graphics and Applications **14** (1994), no. 2, 76–79.

[HS90] W. Hibbard and D. Santek, *The VIS-5D system for easy interactive visualization*, IEEE Visualization '90, 1990, pp. 129–134.

[IBM97] *IBM Visualization Data Explorer: User's Reference*, (Yorktown Heights, NY: IBM Corp), 1997.

[Jac91] J. E. Jackson, *A User's Guide to Principal Components Analysis*, New York: Wiley, 1991.

[Jol86] I.T. Jolliffe, *Principal Component Analysis*, New York : Springer-Verlag, 1986.

[KHS98] K. Koperski, J. Han, and N. Stefanovic, *An efficient two-step method for classification of spatial data*, International Symposium on Spatial Data Handling, July 1998, pp. 45–54.

[Kol84] Bernard Kolman, *Introductory Linear Algebra with Applications*, third ed., Macmillian: New York, 1984.

[Las96] Michael J. Laszlo, *Computational Geometry and Computer Graphics in C++*, Prentice Hall, Upper Saddle River, NJ, 1996.

[LE96] Roland E. Larson and Bruce H. Edwards, *Elementary Linear Algebra*, third ed., D.C. Heath and Company, 1996.

[Lin00] P. Lindstrom, *Out-of-core simplification of polygonal models*, SIGGRAPH 2000, 2000, pp. 259–262.

[LPC+00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, *The Digital Michelangelo Project: 3D Scanning of Large Statues*, SIGGRAPH 2000, 2000, pp. 131–144.

[LSS+98] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin, *MAPS: Multiresolution Adaptive Parameterization of Surfaces*, SIGGRAPH '98, 1998, pp. 95–104.

[LT98] P. Lindstrom and G. Turk, *Fast and memory efficient polygonal simplification*, IEEE Visualization '98, 1998, pp. 279–286.

[MDB87] B. H. McCormick, T. A. Defanti, and M. D. Brown, *Visualization in scientific computing—a synopsis*, IEEE Computer Graphics & Applications 7, 1987, pp. 61–70.

[PS01] Gregory Piatetsky-Shapiro, *KD Nuggets*, http://www.kdnuggets.com/, 2001.

[PTVF97] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipies in C: The Art of Scientific Computing 2ed*, Cambridge University Press : New York, 1997.

[RB93] J. Rossignac and P. Borrel, *Multi-resolution 3D approximations for rendering complex scenes*, Geometric Modeling in Computer Graphics (Genova, Italy) (B. Falcidieno and T. Kunii, eds.), Springer Verlag, June 28–July 2 1993, pp. 455–465.

[RD00] R. K. Rew and G. Davis, *NetCDF: An Interface for Scientific Data Acess*, IEEE Computer Graphics and Applications **10** (1900), no. 4, 76–82.

[Ros94] Lawrence J. Rosenblum, *Research issues in scientific visualization*, IEEE Computer Graphics and Applications **14** (1994), no. 2, 61–63.

[RR96] R. Ronfard and J. Rossignac, *Full-range approximation of triangulated polyhedra*, Computer Graphics Forum **15** (1996), no. 3, 67–76.

[Sam90] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Procssing, and GIS*, Addison-Wesley, Reading, MA, 1990.

[SCN+93] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Caroline Paxson, Alan Su, and Jian Wu, *Tioga: A database-oriented visualization tool*, IEEE Visualization '93, 1993, pp. 86–93.

[SR98] P. H. Smith and J. Van Rosendale, *Data and visualization corridors report on the 1998 CVD workshop series (sponsered by DOE and NSF)*, Tech. Report CACR-164, Center for Advanced Computing Research, California Institue of Technology, 1998.

[SSU90] A. Silbershatz, M. Stonebraker, and J. D. Ullman, *The "Lagunita" of the NSF invitation workshop on the future of database systems research*, Tech. Report TR-90-22, Department of Computer Science, University of Texas at Austin, 1990.

[SZL92] William Schroeder, Jonathan Zarge, and William Lorensen, *Decimation of triangle meshes*, SIGGRAPH '92, 1992, pp. 65–70.

[Tra01] *Transcript of NCVHS 50th Anniversary Symposium*, http://ncvhs.hhs.gov/ncvhs50tr.htm, 2001.

[Tuf93] Edward Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT, 1993.

[Tur92] Greg Turk, *Re-tiling polygonal surfaces*, SIGGRAPH '92, 1992, pp. 55–64.

[Tur96] Greg Turk, *Image-guided streamline placement*, SIGGRAPH '96, 1996, pp. 453–460.

[vDFL+00] Andries van Dam, Andrew S. Forsberg, David H. Laidlaw, Joseph J. LaViola Jr., and Rosemary M. Simpson, *Immersive VR for scientific visualization: A progress report*, IEEE Computer Graphics and Applications **20** (November/December 2000), no. 6, 26–52.

[vW91] Jarke van Wijk, *Spot noise, texture synthesis for data visualization*, SIGGRAPH '91, 1991, pp. 309–318.

[Wat00] Alan Watt, *3D Computer Graphics*, Addision-Wesley, 2000.

[WEL+00] Christopher Weigle, William Emigh, Geniva Liu, Russell Tayler II, James Enns, and Christopher Healey, *Oriented texture slivers: A technique for local value estimation of multiple scalar fields*, Graphics Interface, 2000, pp. 163–170.

Appendix A

Simplification Tables

The following tables are the results of our error calculations performed on the CRU weather dataset. Each table highlights a different simplification method. Within each table are two entries per simplification level. The first entry corresponds to those simplifications where the original implementation was used. The following row is the corresponding simplification with our principal components extension. At each simplification level, we report the various statistics sampled for the mesh.[1] They are the as follows: geometric and attribute average L_2^2, maximum L_1, maximum L_∞, and average L_∞ norms.

As it can be seen in the tables, there is only a small deviation using our extension versus without the extension. In some cases, the maximum error did not change even after using principal components analysis. In these cases, we can be assured that our extensions are performing no worse than without our extensions, in the worst possible case. In general we expect this to be true for early simplification. That is, the simplification algorithm will pick those elements within the low variant regions of the most variant components first (recalling that these most variant components are on top of some attributes with low variation).

Also included in this chapter is the correlation matrix computed for principal components computation. Cross correlation between attributes i and j can be examined by looking in the i^{th}, j^{th} entry in the matrix.

[1] These statistics can be found in Chapter 5.

HQEM Simplification Error					
%Simp	Geom	Attribute Error			
	L_2^2 (10^{-5})	L_2^2 Avg (10^{-2})	L_1 Max	L_∞ Max	L_∞ Avg
90	0.155423	0.31245	0.074884	0.270412	0.028693
90	0.15023	0.35805	0.096763	0.352839	0.030285
80	0.060784	0.05868	0.055194	0.194968	0.011155
80	0.061472	0.06679	0.055194	0.194968	0.01191
70	0.035337	0.02503	0.056853	0.181687	0.006787
70	0.036409	0.03007	0.058720	0.192261	0.007386
60	0.021629	0.01267	0.020176	0.067686	0.004439
60	0.023654	0.01589	0.027455	0.108022	0.004973
50	0.012842	0.00666	0.01193	0.053568	0.002907
50	0.012986	0.00899	0.016695	0.091480	0.003397
40	0.007579	0.00348	0.007844	0.029054	0.001867
40	0.007763	0.00516	0.016695	0.058378	0.002285
30	0.004269	0.00176	0.007779	0.022599	0.001139
30	0.003941	0.00281	0.009592	0.041653	0.001455
20	0.001759	0.00075	0.003801	0.013889	0.000612
20	0.001659	0.00141	0.007779	0.041660	0.000832
10	0.000484	0.00021	0.003121	0.012071	0.000233
10	0.000506	0.00048	0.004943	0.022590	0.000343

Table A.1: HQEM mesh error with and without PCA.

GQEM Simplification Error					
%Simp	Geom	Attribute Error			
	L_2^2 (10^{-5})	L_2^2 Avg (10^{-2})	L_1 Max	L_∞ Max	L_∞ Avg
90	0.110945	0.21984	0.092189	0.343935	0.022532
90	0.111266	0.24180	0.097709	0.270993	0.024581
80	0.054837	0.07710	0.061577	0.247917	0.011975
80	0.058608	0.08761	0.070803	0.222974	0.012981
70	0.034267	0.03812	0.049967	0.247917	0.007759
70	0.038016	0.04764	0.070803	0.189554	0.008609
60	0.022598	0.02119	0.033396	0.135294	0.005310
60	0.024407	0.02874	0.070803	0.195856	0.006010
50	0.014160	0.01219	0.030606	0.120494	0.003605
50	0.015183	0.01691	0.037956	0.195856	0.004177
40	0.008292	0.00701	0.030709	0.120494	0.002384
40	0.009440	0.00981	0.035907	0.125108	0.002828
30	0.004340	0.00343	0.022123	0.069237	0.001454
30	0.005426	0.00544	0.034768	0.120494	0.001786
20	0.001841	0.00143	0.022123	0.053588	0.000774
20	0.002668	0.00266	0.034768	0.120494	0.000999
10	0.000599	0.00044	0.022123	0.052793	0.000306
10	0.000665	0.00082	0.011100	0.052793	0.000414

Table A.2: GQEM mesh error with and without PCA.

O_GQEM Simplification Error					
%Simp	Geom	Attribute Error			
	L_2^2 (10^{-5})	L_2^2 Avg (10^{-2})	L_1 Max	L_∞ Max	L_∞ Avg
90	0.567935	0.29612	0.089121	0.287965	0.026350
90	0.103351	0.22801	0.097709	0.264337	0.023850
80	0.085292	0.08291	0.061577	0.230395	0.012851
80	0.058657	0.08740	0.070803	0.222974	0.013038
70	0.034033	0.03814	0.049967	0.199995	0.007911
70	0.037709	0.04816	0.070803	0.189554	0.008700
60	0.022712	0.02129	0.037956	0.195856	0.005378
60	0.024415	0.02850	0.061577	0.252785	0.006071
50	0.013868	0.01245	0.037956	0.195856	0.003651
50	0.015063	0.01739	0.052071	0.252785	0.004230
40	0.008269	0.00734	0.037956	0.195856	0.002421
40	0.009326	0.01039	0.037956	0.195856	0.002891
30	0.004339	0.00376	0.037956	0.195856	0.001489
30	0.005435	0.00575	0.037956	0.195856	0.001818
20	0.001818	0.00147	0.022123	0.053588	0.000797
20	0.002646	0.00293	0.037956	0.195856	0.001030
10	0.000598	0.00045	0.022123	0.052793	0.000317
10	0.000668	0.00108	0.037956	0.195856	0.000433

Table A.3: O_GQEM mesh error with and without PCA.

PM Simplification Error					
%Simp	Geom	Attribute Error			
	L_2^2 (10^{-5})	L_2^2 Avg (10^{-2})	L_1 Max	L_∞ Max	L_∞ Avg
90	66.3682	0.64650	0.214361	0.84	0.028950
90	35.144	0.58827	0.214361	0.650602	0.029876
80	14.8202	0.29279	0.190984	0.84	0.015152
80	20.5815	0.34396	0.214361	0.650602	0.016528
70	9.10524	0.08213	0.176701	0.390071	0.008425
70	12.2894	0.14542	0.176701	0.390071	0.010269
60	6.22341	0.02751	0.176701	0.390071	0.004960
60	10.1949	0.11341	0.176701	0.390071	0.007299
50	5.60382	0.01703	0.176701	0.390071	0.003161
50	8.34806	0.07752	0.176701	0.390071	0.005102
40	5.27491	0.01329	0.176701	0.390071	0.002076
40	5.39937	0.01949	0.176701	0.390071	0.002722
30	4.98436	0.01102	0.176701	0.390071	0.001288
30	5.00995	0.01304	0.176701	0.390071	0.001667
20	4.72439	0.00958	0.176701	0.390071	0.000729
20	4.72482	0.01029	0.176701	0.390071	0.000968
10	4.49111	0.00866	0.176701	0.390071	0.000341
10	4.49115	0.00900	0.176701	0.390071	0.000475

Table A.4: PM mesh error with and without PCA.

	1	2	3	4	5	6	7	8	9	10	11
1	1	0.436	0.833	-0.13	-0.83	0.570	0.221	0.997	0.996	0.932	-0.00
2	0.436	1	0.372	-0.09	-0.33	-0.13	0.447	0.402	0.471	0.254	0.606
3	0.833	0.372	1	-0.08	-0.92	0.297	-0.04	0.815	0.848	0.827	-0.06
4	-0.13	-0.09	-0.08	1	0.122	-0.27	-0.17	-0.15	-0.11	-0.09	-0.13
5	-0.83	-0.33	-0.92	0.122	1	-0.38	0.129	-0.82	-0.84	-0.83	0.187
6	0.570	-0.13	0.297	-0.27	-0.38	1	-0.09	0.627	0.503	0.698	-0.35
7	0.221	0.447	-0.04	-0.17	0.129	-0.09	1	0.201	0.242	-0.04	0.620
8	0.997	0.402	0.815	-0.15	-0.82	0.627	0.201	1	0.988	0.944	-0.03
9	0.996	0.471	0.848	-0.11	-0.84	0.503	0.242	0.988	1	0.913	0.030
10	0.932	0.254	0.827	-0.09	-0.83	0.698	-0.04	0.944	0.913	1	-0.16
11	-0.00	0.606	-0.06	-0.13	0.187	-0.35	0.620	-0.03	0.030	-0.16	1

Table A.5: Correlation matrix of North America CRU dataset.