SIMPLIFICATION OF TETRAHEDRAL MESHES BY SCALAR VALUE ASSIGNMENT

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Emre Can Sezer

September, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Cevdet Aykanat(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay(Co-supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Bülent Özgüç

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

SIMPLIFICATION OF TETRAHEDRAL MESHES BY SCALAR VALUE ASSIGNMENT

Emre Can Sezer
M.S. in Computer Engineering
Supervisors: Prof. Dr. Cevdet Aykanat and
Assist. Prof. Dr. Uğur Güdükbay
September, 2002

A new approach to simplification of volumetric data over an unstructured tetrahedral mesh is presented. The data consist of sample values of a scalar field defined over a spatial domain, which is subdivided with a tetrahedral mesh. Simplification is performed by means of contraction of the tetrahedra and also of the edges. The simplification algorithm can provide a continuum of aproximate models of the given dataset with any desired degree of accuracy. Hence, the simplification method is suitable for multi-resolution modeling.

The novelty of the approach comes from the arbitrariness in the selection of the point to which a tetrahedron or an edge is contracted. Unlike most of the existing methods, the final vertex of the contraction need not be a vertex of the original mesh. The scalar value to be assigned to the final vertex of contraction is determined by an extremely simple method (both conceptually and computationally), which also provides an estimate of the error of simplification.

The proposed method is applied to two volumetric grids to illustrate its effectiveness in simplification of volumetric data.

Keywords: Volumetric dataset, tetrahedral mesh, tetrahedron contraction, edge contraction, scalar value assignment.

ÖZET

SKALAR DEĞER ATAMA YÖNTEMİ İLE HACİMSEL VERİLERİN YALINLAŞTIRILMASI

Emre Can Sezer
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticileri: Prof. Dr. Cevdet Aykanat and
Assist. Prof. Dr. Uğur Güdükbay
Eylül, 2002

Düzensiz dörtyüzlü ızgaralar üzerinde tanımlanmış hacimsel verilerin yalınlaştırılması için yeni bir yöntem sunulmaktadır. Veri, bir hacim üzerinde tanımlı skalar alandan alınmış örnek değerler ve bu hacimi bölen dörtyüzlüsel ızgaradan oluşur. Yalınlaştırma, dörtyüzlülerin ya da kenarlarının noktaya indirgenmesiyle sağlanır. Yalınlaştırma metodu, istenilen derecedeki doğrulukta modeller sağlayabilir. Dolayısıyla, bu metod çoklu-çözünürlük modellemesi için uygundur.

Metodun güzelliği, dörtyüzlünün ya da kenarının indirgeneceği noktanın değişken olabilmesinden kaynaklanır. Diğer metodların aksine, indirgenilecek noktanın asıl ızgaranın bir köşesi olması gerekmez. İndirgeme sonunda indirgenilen noktanın skalar değeri son derece basit bir yöntemle belirlenir ve bu yöntem hatanın tahmini bir değerini de sağlar.

Önerilen metod, hacimsel verilerin yalınlaştırılmasındaki verimliliğinin görülmesi açısından, iki farklı hacimsel veri örneğinde denenmiştir.

Anahtar sözcükler: Hacimsel veri, dörtyüzlüsel ağ, dörtyüzlü indirgeme, kenar indirgeme, skalar değer atama.

Acknowledgement

I acknowledge my supervisor Dr. Cevdet Aykanat and my co-supervisor Dr. Uğur Güdükbay for their help they extended throughout this study. Special thanks are due to the faculty of the Computer Engineering Department and especially to Dr. Mehmet Baray for giving me the opportunity to do graduate study in a field different from my undergraduate major.

Special thanks to my family for thier endless support and especially to my father for guiding me through and walking along with me in this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A problem commonly encountered in computer graphics is rendering large volumetric datasets. Especially, when the visualization needs to be at interactive speeds the problem is even more severe. This thesis study concentrates on the simplification of volumetric datasets. Volumetric data come from a variety of resources such as medical diagnosis tools or finite element simulations. In either case, some degree of simplification may be required for various reasons, such as reducing visualization times or reducing storage requirements.

A volumetric dataset consists of a set of sample points in R^3 spanning the domain Ω, hence representing a scalar field over Ω (e.g. temperature distribution in a room). Moreover, Ω is spatially subdivided into cells defined as the mesh. A commonly used mesh type is the tetrahedral mesh that is also used in this thesis. It is a type of unstructured triangular volumetric data where the cells are tetrahedra. A tetrahedron consists of 4 triangular faces.

Simplification refers to reducing the number of vertices and tetrahedra while preserving the appearance of the dataset as much as possible. Vertex clustering, decimation and edge contraction are three very commonly used simplification methods. This thesis study concentrates on edge contraction and its extensions to tetrahedron contraction.

An edge contraction operation is the merging of the two vertices of the edge into a single vertex. During this process all tetrahedra that share the edge are contracted to surfaces. If one of the vertices is merged onto the other, the method is called subset placement. However, it is also possible to determine a final vertex position other than the two original vertices. In fact, this has been used in order to reduce the error introduced by the contraction.

Every contraction (simplification) is bound to introduce some error into the simplified model. An accurate error evaluation is very time consuming and is usually done as a post processing step. However, it is desirable to at least approximate the error during simplification. This allows the simplification to terminate when an error threshold is reached.

The problem of simplifying surface datasets have been heavily investigated and many error prediction methods were introduced. Some of these methods have also been applied to volumetric datasets. Moreover, there have been researches that optimize the final vertex position in order to minimize the predicted error.

None of the previous researches on volumetric data simplification attempts to assign a new scalar value to the final vertex after the contraction. In this thesis study, a new method is introduced for the determination of the scalar value of the final vertex, called *Scalar Value Assignment (SVA)*. Moreover, in conjunction with the SVA method an error prediction method is introduced.

The SVA and error prediction methods were applied both with edge contraction and tetrahedron contraction methods. Tetrahedron contraction is the merging of the four vertices of the tetrahedron into a single vertex.

Chapter 2

Background and Related Work

In this chapter we introduce the terminology and basic definitions used throughout this thesis and review previous work directly related to our study. Most of the definitions are standard, and therefore, not referenced individually. Details can be found in [6, 14].

2.1 Meshes and Simplicial Complexes

A *mesh* is a general subdivision of a spatial domain into a finite number of smaller units called *cells*. Symbolically, a mesh in a k-dimensional space is described as

$$\Sigma = \{ \sigma_1, \sigma_2, \ldots, \sigma_m \,|\, \sigma_i \subset \mathbf{R}^k \}$$

The cells σ_i are closed sets that are either disjoint or intersect at their boundaries. Their union

$$\Omega_\Sigma = \bigcup_{i=1}^{m} \sigma_i \subset \mathbf{R}^k$$

constitutes the *domain* of the mesh. The domain is usually a polyhedron, which may be convex or non-convex, or even have cavities. Each cell is uniquely characterized as a convex combination of a set of points in Ω_Σ, called the *vertices* of the cell. Vertices of all cells make up a finite vertex set

$$\mathcal{V} = \{ \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \,|\, \mathbf{v}_i \in \mathbf{R}^k \}$$

Meshes can be structured (Cartesian, regular, rectilinear or curvilinear) or unstructured (regular or irregular). A structured mesh is implicitly defined by its vertices. For unstructured meshes, the mesh topology (the connectivity information of the cells) must be explicitly specified. This introduces inefficiencies in storage, however, they are much more flexible than structured meshes. A commonly used unstructured mesh type is *tetrahedral mesh*, which is well suited for a number of visualization techniques [9, 20, 22], and is described below in a general framework.

Let $\mathcal{V}_\sigma = \{ \mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_d \}$ be a set of $d+1$ affinely independent points in $\mathbf{R}^k$, where $d \leq k$. (By affine independence of the $d+1$ points $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_d$, we mean linear independence of the d vectors $\mathbf{v}_1 - \mathbf{v}_0, \ldots, \mathbf{v}_d - \mathbf{v}_0$ in linear algebraic sense. Thus, any three points not on a straight line, any four points not on a plane, and in general, any $d+1$ points not on a d-dimensional hyperplane are affinely independent.) The convex hull of $\mathcal{V}_\sigma$

$$\sigma = \{ \mathbf{p} = \sum_{i=0}^{d} c_i \mathbf{v}_i \mid \sum_{i=0}^{d} c_i = 1, 0 \leq c_i \leq 1 \}$$

is called a *d-simplex* in $\mathbf{R}^k$. d is referred to as the *order* of σ, and the points in $\mathcal{V}_\sigma$ are called the *vertices* of σ. Any s-simplex formed by a proper subset of $\mathcal{V}_\sigma$ is called an *s-face* of σ. Thus a triangle is a 2-simplex with each edge being a 1-simplex (a 1-face), and a tetrahedron is a 3-simplex with each triangular face being a 2-simplex (a 2-face) and each edge a 1-simplex (a 1-face).

A finite collection Σ of d-simplexes in $\mathbf{R}^k$ that obey the following rules is called a *d-simplicial complex*:

- All faces of a simplex $\sigma \in \Sigma$ belong to Σ.
- For a pair of simplexes $\sigma, \tau \in \Sigma$, either $\sigma \cap \tau = \emptyset$ or $\sigma \cap \tau$ is a proper face of both σ and τ.
- d is the maximum of the orders of simplexes in Σ.

Thus a 2-simplicial complex in $\mathbf{R}^3$ defines a *triangulated* surface (not necessarily a plane), and a 3-simplicial complex defines *tetrahedrized* volume. Clearly,

a simplical complex is a special mesh. In what follows, we will deal mostly with tetrahedral meshes. According to the rules above, a *tetrahedral mesh* is a collection of tetrahedra such that any two tetrahedra are either disjoint or they meet at a common vertex or an edge or a face. Subdivision of a domain with a tetrahedral mesh is called *tetrahedrization*. It should be pointed out that a domain with a vertex set $\mathcal{V}$ may be tetrahedrized in many different ways. *Delanuay tetrahedrization* has some nice properties, which make it suitable for many applications [12]. A Delanuay d-simplicial complex Σ is one in which the interior of the hypersphere circumscribing any d-simplex does not contain a vertex of Σ.

2.2 Volumetric Datasets

Roughly speaking, a *volumetric dataset* is a discrete representation of a scalar field ψ defined over some spatial domain $\Omega \subset \mathbf{R}^3$. Let $\mathcal{D} = \{ \mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n \,|\, \mathbf{d}_i \in \Omega \}$ be the set of discretization points, and $\mathcal{S} = \{ s_1, s_2, \ldots, s_n \,|\, s_i = \psi(\mathbf{d}_i) \}$ be the set of corresponding sample values of ψ. For the discrete representation of ψ to be useful (e.g., for visualization) ψ should be reconstructible at every point in its domain from the sample values with simple and fast calculations. For this purpose, the domain Ω is subdivided with a mesh $\Sigma = \{ \sigma_1, \sigma_2, \ldots, \sigma_m \,|\, \sigma_i \subset \Omega \}$ such that the domain of the mesh

$$\Omega_\Sigma = \bigcup_{i=1}^{m} \sigma_i$$

approximates Ω with any desired degree of accuracy. In general, the vertices of the mesh Σ may or may not coincide with the data points, and a cell may contain none, one or more data points in its interior. For convenience, however, the mesh is usually constructed such that its vertex set $\mathcal{V}$ coincides with $\mathcal{D}$ and no cell contains any other data point in its interior. Such a mesh corresponds to a *maximal subdivision* of Ω. Each cell $\sigma_i \in \Sigma$ is associated with an interpolating function ψ_i that serves to approximate ψ within σ_i, that is, $\psi(\mathbf{p}) \approx \psi_i(\mathbf{p})$ at any $\mathbf{p} \in \sigma_i$. In most cases, ψ_i are linear interpolants of ψ that can be generated from the values of ψ at the vertices of Σ, which are either part of (when $\mathcal{V} \subset \mathcal{D}$) or interpolated from the given data. The mesh Σ, the set of interpolants $\Psi_\Sigma =$

$\{ \psi_1, \ldots, \psi_m \}$, data points $\mathcal{D}$ and the sample values $\mathcal{S}$ altogether constitute a volumetric dataset.[1]

Maximal tetrahedral meshes allow for easy construction of the interpolants ψ_i associated with the tetrahedra σ_i as linear functionals. Suppose that for a point $\mathbf{p}$ described in homogeneous coordinates as

$$\mathbf{p} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{2.1}$$

the function ψ_i has the value

$$\psi_i(\mathbf{p}) = \boldsymbol{\alpha}_i^T \mathbf{p} \tag{2.2}$$

where $\boldsymbol{\alpha}^T$ is a constant row 4-vector. In open form, the expression for $\psi_i(\mathbf{p})$ is

$$\psi_i(\mathbf{p}) = \alpha_{ix} x_p + \alpha_{iy} y_p + \alpha_{iz} z_p + \alpha_{ic}$$

Representing $\mathbf{p}$ in homogeneous coordinates allows for including the constant term α_{ic} into the parameter vector $\boldsymbol{\alpha}_i$. Evaluating ψ_i at the vertices of σ_i we obtain

$$\boldsymbol{\alpha}_i^T \begin{bmatrix} \mathbf{v}_{i0} & \mathbf{v}_{i1} & \mathbf{v}_{i2} & \mathbf{v}_{i3} \end{bmatrix} = \begin{bmatrix} s_{i0} & s_{i1} & s_{i2} & s_{i3} \end{bmatrix}$$

which can be written in compact form as

$$\boldsymbol{\alpha}_i^T V_i = \mathbf{s}_i^T \tag{2.3}$$

The fact that vertices of σ_i are not planar guarantees that the matrix V_i in (2.3) is non-singular. Hence, $\boldsymbol{\alpha}_i^T$ can be solved uniquely as

$$\boldsymbol{\alpha}_i = \mathbf{s}_i^T V_i^{-1} \tag{2.4}$$

and the linear interpolant of ψ for σ_i is obtained as

$$\psi_i(\mathbf{p}) = \mathbf{s}_i^T V_i^{-1} \mathbf{p} \tag{2.5}$$

[1]Scientific datasets are generally multi-valued. That is, for each vertex there may be a number of scalar values associated with it, each being a sample of a different scalar field (e.g., temperature, pressure, velocity, etc.) over the same domain. However, usually each of these fields is studied independently.

The expression for $\psi_i(\mathbf{p})$ above has an interesting interpretation. If, for a fixed $\mathbf{p}$, we define

$$V_i^{-1}\mathbf{p} = \mathbf{c} = \begin{bmatrix} c_{i0} \\ c_{i1} \\ c_{i2} \\ c_{i3} \end{bmatrix}$$

then

$$\mathbf{p} = V_i\mathbf{c}_i = c_{i0}\mathbf{v}_{i0} + c_{i1}\mathbf{v}_{i1} + c_{i2}\mathbf{v}_{i2} + c_{i3}\mathbf{v}_{i3} \tag{2.6}$$

which characterizes $\mathbf{p}$ as a linear combination of the vertices of σ_i. Considering the last components of both sides of (2.6), we observe that the coefficients satisfy

$$c_{i0} + c_{i1} + c_{i2} + c_{i3} = 1$$

If, in addition, $c_i \geq 0$, then the right-hand side of (2.6) specifies $\mathbf{p}$ as a convex combination of the vertices of σ_i, implying that $\mathbf{p} \in \sigma_i$. Then, (2.5), written in terms of the coefficients in $\mathbf{c}_i$ in open form as

$$\psi_i(\mathbf{p}) = \mathbf{s}_i^T\mathbf{c}_i = c_{i0}s_{i0} + c_{i1}s_{i1} + c_{i2}s_{i2} + c_{i3}s_{i3} \tag{2.7}$$

specifies $\psi_i(\mathbf{p})$ as the same linear combination of the associated scalar values.

2.3 Mesh Simplification

Consider a mesh Σ defined over a vertex set $\mathcal{V}$ and covering a domain Ω_Σ. The mesh may or may not be associated with a scalar field as, for example, in the case of a volumetric dataset, or in the case of a triangulated surface of a 3D object. Let us call Σ, together with the sample values of a scalar field at the vertices or its interpolants associated with the cells (if any such field accompanies Σ), a *reference model* [3]. For various reasons as described below, it may be desirable to obtain an *approximate model* consisting of a mesh Σ' with fewer cells defined over a vertex set $\mathcal{V}'$ with fewer vertices such that the domain of Σ' is approximately the same as that of Σ ($\Omega_{\Sigma'} \approx \Omega_\Sigma$). If a scalar field ψ is associated with Σ, it is also required that the fields reconstructed from Σ' and Σ should be close ($\psi_{\Sigma'} \approx \psi_\Sigma$).

The process of obtaining an approximate mesh from a reference mesh is called *simplification*. Simplification may be performed for various reasons, such as eliminating redundancies in the model introduced during its generation, or reducing the model size to satisfy storage constraints, or improving runtime performance of an application working with the model [14]. In either case, the purpose of simplification is to reduce the amount of data without sacrificing useful information. Usually, these two objectives are conflicting, and depending on the particular application, one is favored at the expense of the other.

The main parameters in a typical simplification are a cell (polygon or polyhedron) count and/or an error threshold. Cell count measures the compactness of the approximate model and the error parameter measures the quality of the approximation. The simplification error can be defined in a variety of ways depending on the nature of the model. In the case of rendering 2D surfaces of 3D objects, the error is based on geometric closeness of $\Omega_{\Sigma'}$ and Ω_{Σ}, for a faithful representation of the object is the main concern. In volumetric datasets, closeness of the reconstructed fields is even more important. Cignoni et al. [1] distinguish these two types of errors as the *domain error* and the *field error*. A convex volumetric dataset has a relatively small number of vertices on its boundary, and therefore, the domain error due to simplification is usually ignored. In fact, a simplification algorithm may be designed to leave the boundary unchanged, resulting in zero domain error.

Many applications use a number of approximations of a given reference mesh interchangeably. For example, in computer graphics distant objects need not be modeled in as much detail as close objects. Availability of several versions of the same object, each at a different level of detail, allows a rendering algorithm to choose the coarsest satisfactory model, and thus increases the rendering speed. Such a sequence of approximations is called a *level of detail* (LOD) representation of a given reference mesh. LOD representations are usually obtained by iterative application of a specific simplification algorithm, although it is also possible to obtain each approximation directly from the reference mesh. Because LODs are computed off line, such an approach to simplification is also called *static simplification* [14]. In a LOD representation, the whole dataset is stored for every

approximation. In addition to storage requirements, LOD representations have the disadvantage that they do not allow for a smooth transition between models.

An alternative to static LOD representation is dynamic *multi-resolution modeling*, which is based on the idea of recording a series of simple operations that allow for reconstruction of an approximate model with any desired degree of accuracy. The desired model may be obtained by simplifying the reference model until a specified error threshold is reached, or alternatively, by refining a *base model* (the coarsest approximation with the largest error) until the error drops below the specified threshold. In either case, the transformations (edge contraction or vertex split) rather than the individual approximate models are stored. These operations are usually local and are stored in tree structures. In dynamic simplifications the desired model is formed in real time (by performing the simplification or refinement operations) [14]. Advantages of dynamic simplification are better granularity (much greater number of representations of a reference model), efficiency in storage space, and smooth transition between models. The surveys by Garland [6] and Puppo and Scopigno [16] explain different algorithms of creating multi-resolution models and their data structures in detail.

Polygonal simplification algorithms are also classified according to the mechanisms they employ as *clustering*, *decimation* and *contraction* algorithms. Clustering algorithms are based on replacing a group (cluster) of vertices in a mesh by a single representative vertex. Decimation algorithms iteratively remove vertices or higher dimensional units from a mesh. Contraction algorithms collapse edges, faces or cells of a mesh to vertices, thereby eliminating a number of cells at a single operation. These algorithms are discussed in the next section.

2.4 Related Work

Most of the research on mesh simplification has been concentrated on surface meshes [4, 7, 11, 14, 15], with relatively few of the methods having also been extended to simplification of volumetric data. In this section, we present a brief

outline of some of the basic simplification methods.

One of the earliest work on vertex clustering algorithms was proposed by Rossignac and Borrel [18]. In their method, a regular grid is superimposed on the domain of the mesh. Vertices in the same cell of the grid are then collapsed to the single representative vertex within the cell, and the mesh topology is updated accordingly (Figure 2.1). The representative vertex is chosen to be the most important vertex in the cell, where importance of the vertices is determined according to their curvature and the size of the faces they are attached to. Quality of the approximation depends on the resolution of the grid as well as the regularity of the mesh. Although this method is very fast, it is not adaptive, meaning that simplification occurs at the same level over the entire mesh. Low and Tan [13] improved the method of [18] by introducing the floating-cell algorithm. In their method, a cell of specified size is centered on the most important vertex, all the vertices within the cell are merged into the center vertex, the mesh is modified accordingly, and the process is repeated with the next most important cell. The algorithms of [18] and [13] as well as their variations developed later are very fast and can be applied to arbitrary polygonal meshes. However, since the vertex clustering algorithms do not preserve topology of the mesh, the quality of the resulting meshes are poor. In [7], it is pointed out that clustering methods can be generalized to use an adaptive grid structure such as an octree [15] to improve quality.

Vertex decimation methods, which are based on iterative removal of vertices of a mesh, have been used mostly on triangulated surfaces [4, 14, 19]. In the basic method of Schroeder et al. [19], a vertex is chosen for removal, all the edges connected to the chosen vertex are deleted, and the resulting hole is then re-triangulated (Figure 2.2). Due to the re-triangulation process the method is restricted to manifold surfaces only[2]. However, it provides reasonable efficiency and quality together. Extension of decimation algorithms to volumetric meshes has been considered by Renze and Oliver [17]. The problem with volumetric

[2]A manifold surface is a 2D mesh in which the neighborhood of every vertex, edge or a face consists of a connected ring of polygons that form a single surface.

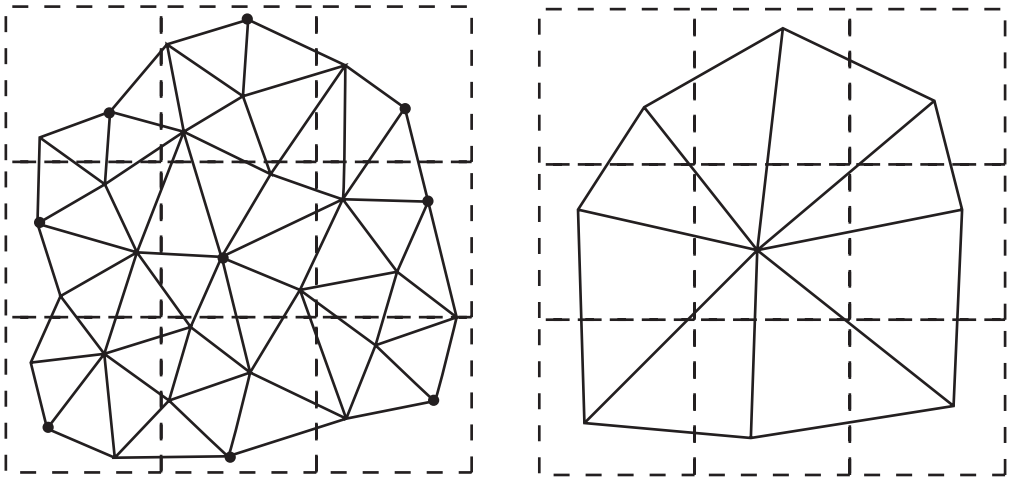

Figure 2.1: Vertex clustering on a 2D mesh

meshes is that the hole after the removal of a vertex may be non-convex, creating difficulties in re-tetrahedrization of the hole. In fact, even the problem of whether a non-convex domain is tetrahedrizable is NP-complete [3]. Therefore, decimation algorithms may not work as effectively for volumetric meshes as they do for surfaces. Decimation methods are not restricted to vertex removal. Cohen et al. [4] described a decimation approach that iteratively removes triangles from a surface mesh. In their approach, they created *simplification envelopes* that consist of two offset surfaces, which are copies of the original surface offset by a fixed amount in both directions along the normals of the vertices. By keeping the surface of the re-triangulated hole after the removal of a triangle, they guarantee that the simplified surface does not deviate from the original by more than the fixed amount of the offset.

Decimation methods lie in the class of *incremental* simplification methods, in which a sequence of local operations produce a sequence of approximate meshes with little differences, thus resulting in good quality multi-resolution models. Edge contraction algorithms, which are widely used in surface simplification, are also in this class. Like decimation algorithms, edge contraction methods iteratively select an edge to contract and remove all faces that include the selected edge. Edges are weighted with respect to the error they will introduce if they

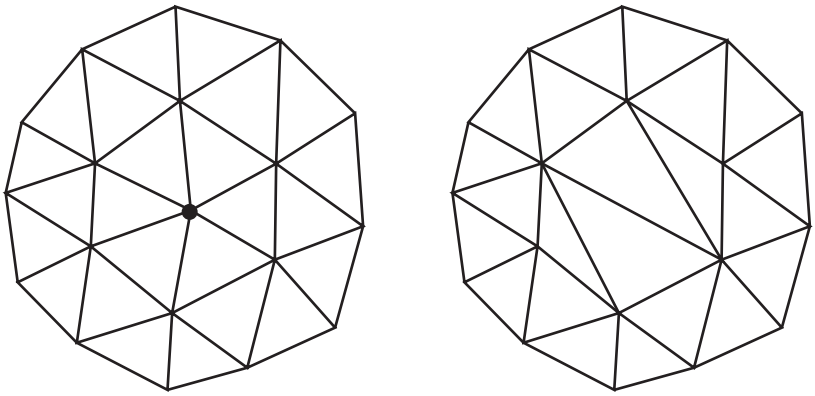

Figure 2.2: Decimation illustrated on a 2D mesh

are collapsed, which is called the *predicted error*. At each iteration, the edge with the smallest predicted error is contracted to a point. The process continues until a specified cell count or an error threshold is reached. There have been many edge contraction algorithms with similar basic operations, the differences stemming from the choice of the position of the new point after contraction, the error metric used, and the way error is estimated.

In order to have a good quality multi-resolution model, the error must increase smoothly through the simplification process. Iterative algorithms are naturally suitable for this purpose. An iterative selection process requires calculation of the predicted errors of all candidates, which would take enormous time if to be done at every iteration. The basic approach to overcome this problem is to perform local operations and to calculate only the predicted errors of those edges that are modified or have modified neighbors. Combined with piecewise linearity used in visualization algorithms, this approach often produces satisfactory results, and is therefore used in a majority of the edge contraction algorithms.

Most edge contraction algorithms choose one of the vertices of the edge to be collapsed as the point of contraction. Garland and Heckbert [8] call this type of edge contraction *subset placement*, because the vertex set of the mesh after each edge contraction is a subset of the vertex set of the previous mesh.

Trotts et al. [21] used the subset placement strategy in the simplification of tetrahedral meshes. In their method, a tetrahedron is collapsed to one of its four vertices by three consecutive edge contractions as illustrated in Figure 2.3 for a triangular surface mesh (where a triangle is collapsed to one of its vertices by two edge contractions). Although the approach seems simple to implement, the order in which the edges are contracted affects both the final point of contraction and the approximation error. To deal with this problem, they simply check all possibilities and determine the sequence that introduces the lowest error. Their error prediction is accurate but complicated (takes to much time to determine the predicted error). They measure the error by the maximum difference between the scalar fields before and after the contraction. To determine the error they check the difference in the scalar field at every intersection of the original and the simplified mesh. Due to linearity of the interpolants of the scalar fields, it turns out that the maximum difference occurs at one of the points at which an edge of one of the new tetrahedra in the simplified mesh intersects a face of one of the tetrahedra of the original mesh. Figure 2.4 illustrates how these points are found on a 2D mesh. Their paper also discusses preservation of the boundary, which is also considered in our study.

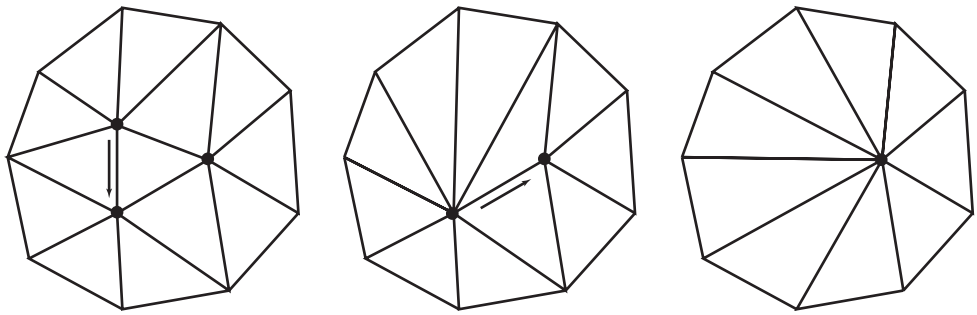

Figure 2.3: Triangle contraction by subset placement

The subset placement strategy has also been used by Cignoni et al. [1] in simplification of tetrahedral meshes, who adopted the *quadric error metric* for calculation of both the domain error and the field error due to the collapse of a tetrahedron. Quadric error metric introduced by Garland and Heckbert [7] is a

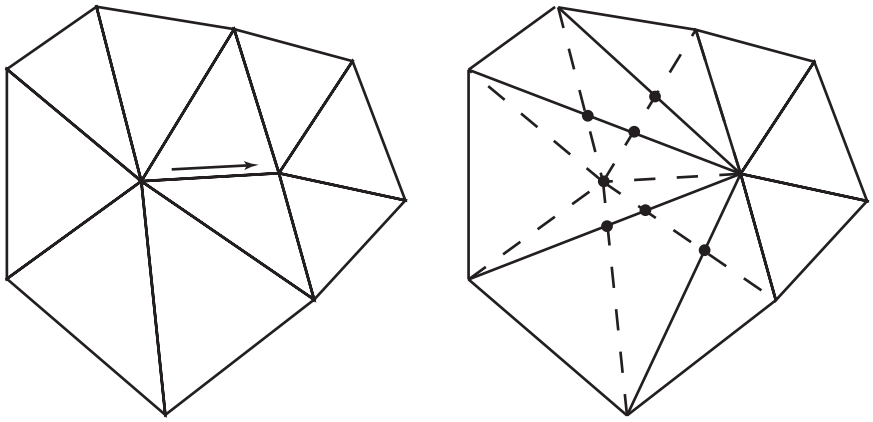

Figure 2.4: Calculation of the error bound in triangle contraction adopted from [21]

clever way of estimating the geometric error resulting from merging two vertices (not necessarily forming an edge) of a surface mesh, and is worth mentioning briefly.

The main idea of the quadric error metric is to associate a set of planes with each vertex, and define the sum of the squared distances of that vertex from the associated planes as the error of the vertex. (Initially, the planes associated with a vertex $\mathbf{v}$ are the faces incident on $\mathbf{v}$, and therefore, the error of $\mathbf{v}$ is zero.) When a source vertex $\mathbf{v}_s$ is merged with a destination vertex $\mathbf{v}_d$, the set of planes associated with $\mathbf{v}_d$ is augmented by the set of planes associated with $\mathbf{v}_s$, and the error of $\mathbf{v}_d$ is updated. The innovative contribution of [7] is to devise a method of updating the error of $\mathbf{v}_d$ without actually keeping track of the planes associated with the vertices. If the equation of the ith plane associated with a vertex $\mathbf{v}$ in homogeneous coordinates is

$$\mathbf{n}_i^T \mathbf{p} = 0 \tag{2.8}$$

In open form (2.8) is written as

$$n_{xi}x_p + n_{yi}y_p + n_{zi}z_p + n_{ci} = 0$$

where n_{xi}, n_{yi} and n_{zi} are the components of the unit normal of the plane, and n_{ci} is the distance from the origin. Expressing $\mathbf{p}$ in homogeneous coordinates as

in (2.1) allows for the constant term n_{ci} to be included as a fourth components of $\mathbf{n}_i^T$. Then the square of the distance of $\mathbf{v}$ from that plane is simply

$$d_i^2 = (\mathbf{n}_i^T \mathbf{v})^2 = \mathbf{v}^T \mathbf{n}_i \mathbf{n}_i^T \mathbf{v}$$

The error of $\mathbf{v}$ is then the sum of the squared distances over all associated planes, that is,

$$e(\mathbf{v}) = \sum_i d_i^2 = \mathbf{v}^T (\sum_i \mathbf{n}_i \mathbf{n}_i^T) \mathbf{v} = \mathbf{v}^T Q \mathbf{v}$$

where Q is the 4×4 quadric error matrix of $\mathbf{v}$. Thus to calculate the error of $\mathbf{v}$ it suffices to know its quadric error matrix (which requires storing only ten coefficients as Q is symmetric). When a vertex $\mathbf{v}_s$ is merged into $\mathbf{v}_d$, the quadric error matrix of $\mathbf{v}_d$ is updated simply as $Q_d \leftarrow Q_d + Q_s$. Although adding Q_s to Q_d results in double counting of the planes common to $\mathbf{v}_s$ and $\mathbf{v}_d$, and thus an overestimation of the error, it is much faster than the more accurate alternative of the inclusion-exclusion method (in which the contribution of the doubly counted planes is later subtracted from the final quadric).

Cignoni et al. [1] estimated the domain error by providing a quadric matrix for each of the boundary vertices. To calculate the field error, they associated with each vertex $\mathbf{v}$ a set of linear interpolants ψ_i as given by (2.1), and defined the field quadric error of a vertex $\mathbf{v}$ as

$$e_f(\mathbf{v}) = \sum_i (\psi_i(\mathbf{v}) - s)^2 = \mathbf{v}^T (\sum_i \boldsymbol{\beta}_i \boldsymbol{\beta}_i^T) \mathbf{v} = \mathbf{v}^T Q_f \mathbf{v}$$

where s is the field on $\mathbf{v}$ and

$$\boldsymbol{\beta}_i = \boldsymbol{\alpha}_i - \begin{bmatrix} 0 \\ 0 \\ 0 \\ s \end{bmatrix}$$

As in the case of domain quadric, initially the fields associated with $\mathbf{v}$ are the linear interpolants of the tetrahedra incident on $\mathbf{v}$ so that $e_f(\mathbf{v}) = 0$. However, when a source vertex $\mathbf{v}_d$ is merged with a destination vertex $\mathbf{v}_s$, then the field quadric matrix of $\mathbf{v}_s$ is updated as $Q_{fs} \leftarrow Q_{fs} + Q_{fd}$.

An improvement over subset placement strategy was proposed by Garland and Heckbert [7], who considered the problem of optimal choice of the final vertex position in association with an edge contraction algorithm in order to minimize the quadric error. Cignoni et al. [1] proposed a sub optimization scheme that selects the best final vertex position among a number of candidates.

Closely related to mesh simplification is refinement, which is the process of adding detail to a coarse representation to enhance the detail level and reduce the approximation error. In refinement, a coarse approximation of the reference mesh is taken as a base mesh, whose vertices form a small subset of the data points. At each iteration, a data point is inserted into the mesh followed by a re-triangulation process to construct a finer mesh in which the newly inserted data point becomes the vertex of a cell. Hamann and Chen [10] have extended this method for volume data. Their method selects a data point based on curvature, and inserts it into the convex hull of the of the domain of the dataset. The mesh is then re-tetrahedralized with local modifications in order to minimize some error criteria. Like in most refinement methods, the base mesh must be convex. Cignoni et al [2] have also proposed a refinement method, which selects the data point that introduces the maximum error with respect to the reference mesh. They used Delanuay tetrahedrization and local operations to modify the mesh when a new point is inserted [12]. Later, they extended their work to include non-convex meshes obtained by the deformation of convex domains (curvilinear datasets) [3]. (The data structure used in the implementation of this study is very similar to the one used in [3].)

Chapter 3

Optimization in Contraction Operations

In this chapter, we consider the problem of contracting a primitive in a volumetric mesh with an associated scalar field (e.g., a volumetric dataset), while introducing the minimum error with respect to an error criteria.

3.1 Formulation of the Optimization Problem

Consider a mesh $\Sigma = \{ \sigma_1, \ldots, \sigma_m \}$ and a set of associated linear interpolating functions $\{ \psi_1, \ldots, \psi_m \}$ that forms a piecewise linear approximate ψ_Σ of a given scalar field ψ. We assume that each ψ_i can be generated uniquely from the scalar values assigned to the vertices of the corresponding cell σ_i, which are either given (if the vertex of interest is also a data point) or assigned (if the vertex is generated by a previous contraction and does not coincide with a data point). Let τ be any simplex that is to be contracted to a vertex. τ need not be a maximal simplex. For example, in a tetrahedral mesh, τ may be a tetrahedron, or a face, or an edge of a tetrahedron.

Let $\Sigma' = \{ \sigma'_1, \ldots, \sigma'_{m'} \}$ denote the mesh after contraction of τ to a new

vertex $\mathbf{v}'$ and a suitable modification of the local topology of τ. Let s' denote the scalar value assigned to $\mathbf{v}'$, and let ψ_i' denote the interpolants (vertices) of σ_i' recalculated from s' and the scalar values of the vertices common to Σ and Σ' (untouched vertices). Then, the optimization problem concerned with the contraction of τ to $\mathbf{v}'$ can be formulated as

$$\min_{\mathbf{v}' \in \tau} \min_{s'} e(\psi_\Sigma, \psi_{\Sigma'})$$

where

$$e(\psi_\Sigma, \psi_{\Sigma'}) = e(\mathbf{v}', s')$$

is a suitable error measure that evaluates the quality of the contraction. The optimization problem is also associated with some constraints such as preservation of the boundary of the domain Ω_Σ of the mesh [5] and avoidance of the flipping of the cells [3].

3.2 A Summary of Solutions

Clearly, solution of the optimization problem depends heavily on the choice of the error measure. In the following, we summarize possible choices of the error measure and the corresponding solutions. For convenience, we will illustrate various approaches on the one-dimensional mesh shown in Figure 3.1, where

$$\mathcal{V} = \{ v_1, v_2, v_3, v_4 \} = \{ 0, 1, 3, 4 \}$$

$$\mathcal{S} = \{ s_1, s_2, s_3, s_4 \} = \{ 0, 1, 2, 1 \}$$

and

$$\Sigma = \{ \sigma_1, \sigma_2, \sigma_3 \}$$

with

$$\begin{aligned} \sigma_1 &= [0, 1] , \qquad \psi_1(x) = x \\ \sigma_2 &= [1, 3] , \qquad \psi_2(x) = \frac{1}{2}\, x + \frac{1}{2} \\ \sigma_3 &= [3, 4] , \qquad \psi_3(x) = -x + 5 \end{aligned}$$

Let the simplex $\tau = \sigma_2$ be contracted to a point $v' \in [1, 3]$, which is assigned the scalar value s'. The mesh after contraction is also shown in Figure 3.1, where

$$\mathcal{V}' = \{ v'_1, v'_2, v'_3 \} = \{ 0, v', 4 \}$$

and

$$\Sigma' = \{ \sigma'_1, \sigma'_2 \}$$

with

$$\sigma'_1 = [0, v'] \,, \qquad \psi'_1(x) = \frac{s'}{v'}\, x$$

$$\sigma'_2 = [v', 4] \,, \qquad \psi'_2(x) = \frac{1 - s'}{4 - v'}\, x + \frac{4s' - v'}{4 - v'}$$

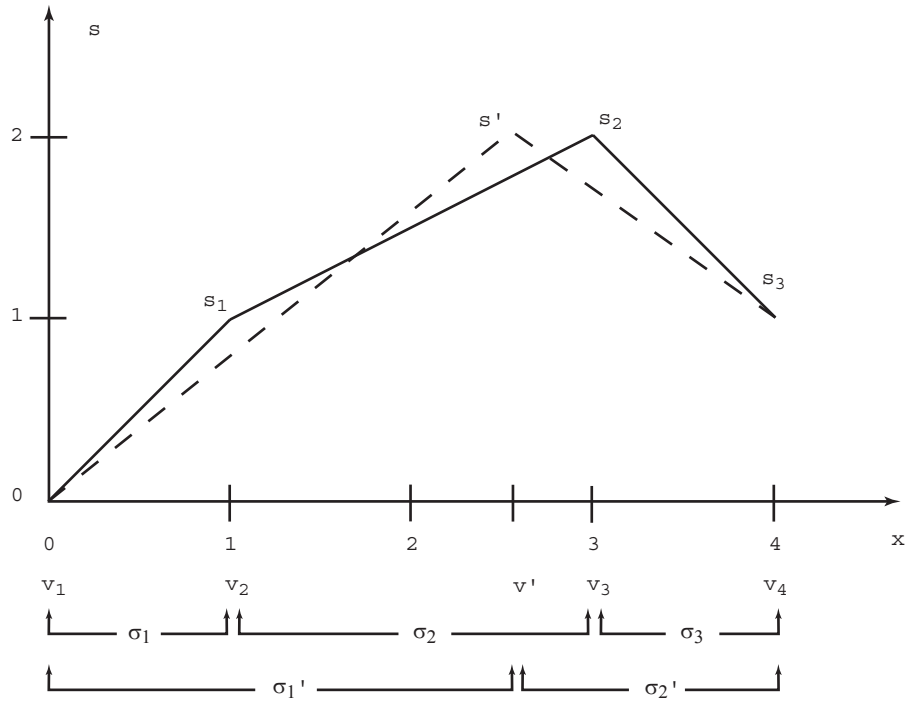

Figure 3.1: A one dimensional mesh and contraction of the edge σ_2. (Original mesh is shown with solid lines and the simplified mesh is shown with dashed lines)

3.2.1 Generalized Quadric Error

Garland and Heckbert [8] proposed a generalized quadric error by augmenting the vertices with the scalar values associated with them. Their approach is equivalent to finding a point in homogeneous coordinates

$$\mathbf{v}' = \begin{bmatrix} v' \\ s' \\ 1 \end{bmatrix}$$

in the xs-plane which minimizes the sum of the quadrics associated with the endpoints

$$\mathbf{v}_2 = \begin{bmatrix} v_2 \\ s_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{v}_3 = \begin{bmatrix} v_3 \\ s_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

Note that this formulation is equivalent to treating the dataset $(\Sigma, \mathcal{S})$ as a one-dimensional mesh (1-simplicial complex) in $\mathbf{R}^2$ whose cells are the lines L_1, L_2, L_3 shown in Figure 3.2.

The quadric associated with $\mathbf{v}_2$ is the sum of the squared distances of a point

$$\mathbf{p} = \begin{bmatrix} x \\ s \\ 1 \end{bmatrix}$$

to the neighboring lines L_1 and L_2, and similarly, the quadric associated with $\mathbf{v}_3$ is the sum of the squared distances of $\mathbf{p}$ to the neighboring lines L_2 and L_3.

From the equations of the lines

$$L_1 : \quad \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} x \\ s \\ 1 \end{bmatrix} = \mathbf{n}_1^T \mathbf{p}$$

$$L_2 : \quad \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} x \\ s \\ 1 \end{bmatrix} = \mathbf{n}_2^T \mathbf{p}$$

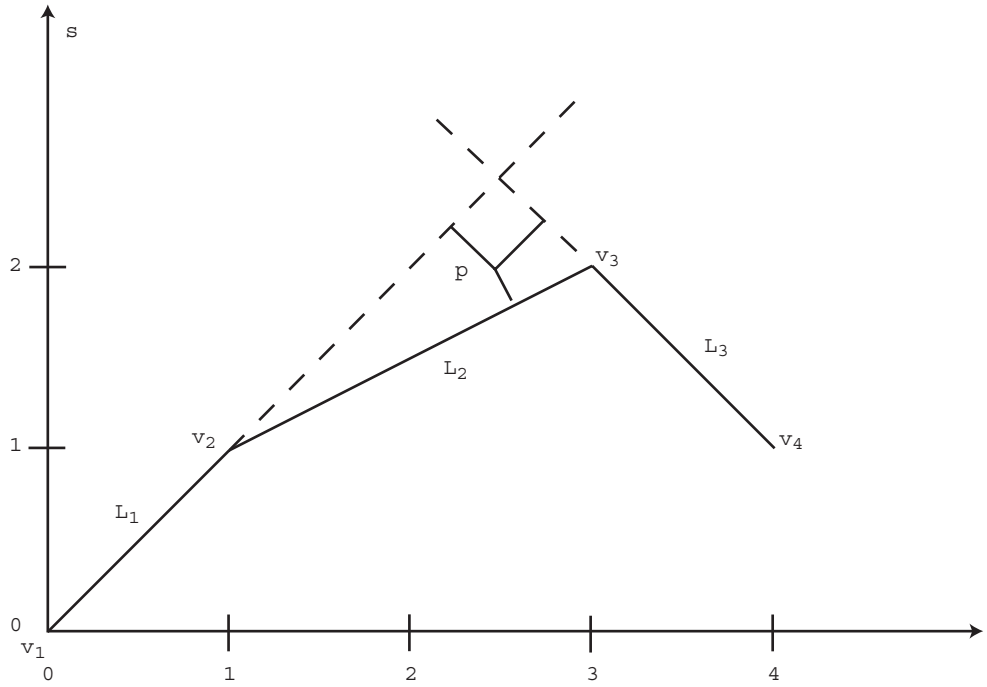

Figure 3.2: The mesh in Figure 3.1 as a mesh in $\mathbf{R}^2$

$$L_3 : \; [\, \frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}} \quad -\frac{5}{\sqrt{2}} \,] \begin{bmatrix} x \\ s \\ 1 \end{bmatrix} \;=\; \mathbf{n}_3^T \mathbf{p}$$

the quadric matrices are formed as

$$Q_2 = \mathbf{n}_1\mathbf{n}_1^T + \mathbf{n}_2\mathbf{n}_2^T = \frac{1}{10} \begin{bmatrix} 7 & -9 & 2 \\ -9 & 13 & -4 \\ 2 & -4 & 2 \end{bmatrix}$$

and

$$Q_3 = \mathbf{n}_2\mathbf{n}_2^T + \mathbf{n}_3\mathbf{n}_3^T = \frac{1}{10} \begin{bmatrix} 7 & 1 & -23 \\ 1 & 13 & -29 \\ -23 & -29 & 127 \end{bmatrix}$$

Consequently, the quadric error associated with point $\mathbf{v}'$ is

$$e(v', s') = Q(\mathbf{v}') \;=\; (\mathbf{v}')^T (Q_2 + Q_3)(\mathbf{v}')$$

$$= \frac{1}{10}\begin{bmatrix} v' & s' & 1 \end{bmatrix} \begin{bmatrix} 14 & -8 & -21 \\ -8 & 26 & -33 \\ -21 & -33 & 129 \end{bmatrix} \begin{bmatrix} v' \\ s' \\ 1 \end{bmatrix}$$

Note that the distance of point $\mathbf{v}'$ to the line L_2 is counted twice, which is a weakness of the quadric error measure.

Minimization of $Q(\mathbf{v}')$ yields

$$\begin{bmatrix} v' \\ s' \end{bmatrix} = \begin{bmatrix} 14 & -8 \\ -8 & 26 \end{bmatrix}^{-1} \begin{bmatrix} 21 \\ 33 \end{bmatrix} = \begin{bmatrix} 2.7 \\ 2.1 \end{bmatrix}$$

The corresponding simplified mesh is shown in Figure 3.3.

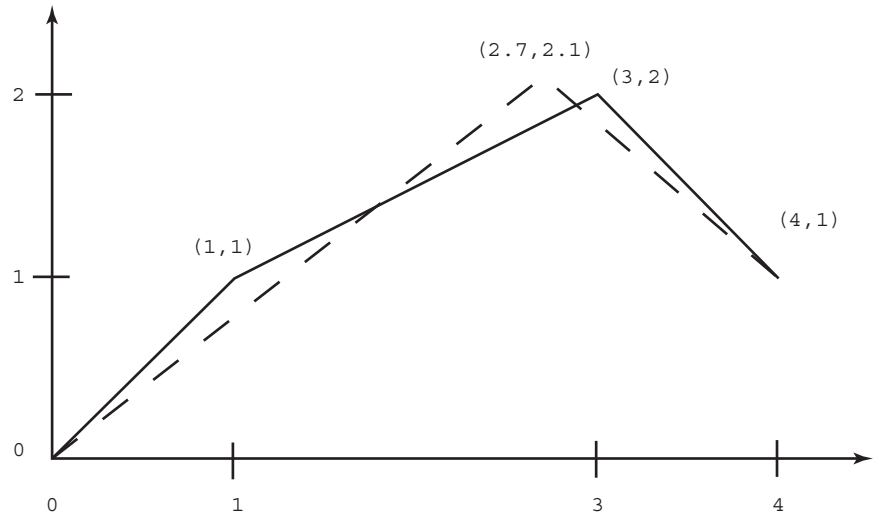

Figure 3.3: Contraction based on generalized quadric error metric

This method of using generalized quadric metric as a measure of error can easily be generalized to higher dimensional meshes with multiple values assigned to the vertices. However, treating the field values as generalized coordinates is debatable for the following reason. If the cell to be contracted is not a boundary cell (as in the illustrative example above), then the domain error due to contraction is of no concern, and only the field error needs to be considered. However,

attachment of the field values to vertices as additional coordinates transforms the problem to one of minimizing the domain error in a higher dimensional space, which in turn, necessarily involves inclusion of the domain error of the original mesh into consideration. This can be better understood by noticing that $Q(\mathbf{v}')$ in the illustrative example measures the domain error introduced by contracting line L_2, which is a boundary edge of the line mesh $\{ L_1, L_2, L_3 \}$, to the point $\mathbf{v}' \in \mathbf{R}^2$.

3.2.2 Field Quadric

Cignoni et al. [1] proposed to use a weighted sum of domain and field error quadrics as an error measure. Although their purpose was to evaluate the error in a subset placement strategy rather than to minimize it, their error measure can also be used for optimum choice of the final vertex position as we illustrate below.

Referring to the illustrative example, since the cell to be contracted is not a boundary cell we need not consider the domain error quadric, and concentrate only on the field error. To calculate the field error, we observe that associated with the vertex v_2 are the linear interpolants ψ_1 and ψ_2, and with v_3 are ψ_2 and ψ_3. Hence, the field error quadric associated with v_2 at a point

$$\mathbf{p} = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

is

$$\begin{aligned} e_2(\mathbf{p}) &= (\psi_1(x) - s_2)^2 + (\psi_2(x) - s_2)^2 \\ &= \begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} \frac{5}{4} & -\frac{5}{4} \\ -\frac{5}{4} & \frac{5}{4} \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \mathbf{p}^T Q_2 \, \mathbf{p} \end{aligned}$$

Similarly, the field error quadric associated with v_3 at $\mathbf{p}$ is

$$\begin{aligned} e_3(\mathbf{p}) &= (\psi_2(x) - s_3)^2 + (\psi_3(x) - s_3)^2 \\ &= \begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} \frac{5}{4} & -\frac{15}{4} \\ -\frac{15}{4} & \frac{45}{4} \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \mathbf{p}^T Q_3 \, \mathbf{p} \end{aligned}$$

Cignoni et al. used these quadrics in evaluating the field errors when either of v_2 or v_3 is merged to the other. If v_3 is merged to v_2, then the quadric of v_2 is updated as

$$Q_2 \leftarrow Q_2 + Q_3 = \begin{bmatrix} \frac{5}{2} & -\frac{10}{2} \\ -\frac{10}{2} & \frac{25}{2} \end{bmatrix}$$

and the field error due to the contraction is

$$e(\mathbf{v}_2) = (\mathbf{v}_2)^T Q_2(\mathbf{v}_2) = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{5}{2} & -\frac{10}{2} \\ -\frac{10}{2} & \frac{25}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 5$$

If, on the other hand, v_2 is merged to v_3, then the quadric of v_3 is updated as

$$Q_3 \leftarrow Q_3 + Q_2 = \begin{bmatrix} \frac{5}{2} & -\frac{10}{2} \\ -\frac{10}{2} & \frac{25}{2} \end{bmatrix}$$

and the field error becomes

$$e(\mathbf{v}_3) = (\mathbf{v}_3)^T Q_3(\mathbf{v}_3) = \begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} \frac{5}{2} & -\frac{10}{2} \\ -\frac{10}{2} & \frac{25}{2} \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} = 5$$

In either case, the scalar value assigned to the final vertex (v_2 or v_3) is the same as the field on that vertex before the contraction. The resulting contractions are shown in Figure 3.4, where $e(\mathbf{v}_2) = d_{32}^2 + d_{33}^2$ and $e(\mathbf{v}_3) = d_{21}^2 + d_{22}^2$.

We can use the quadrics associated with v_2 and v_3 in finding the optimal final vertex position. If v_2 and v_3 are both merged to a new vertex $v' \in [1, 3]$, then the field error at v' is

$$e(\mathbf{v}') = (\mathbf{v}')^T (Q_2 + Q_3)(\mathbf{v}') = \begin{bmatrix} v' & 1 \end{bmatrix} \begin{bmatrix} \frac{5}{2} & -\frac{10}{2} \\ -\frac{10}{2} & \frac{25}{2} \end{bmatrix} \begin{bmatrix} v' \\ 1 \end{bmatrix}$$

Minimization of $e(\mathbf{v}')$ with respect to v' yields

$$v' = 2.0$$

The mesh after contracting σ_2 to v' found above is shown in Figure 3.5, where v' is assigned the value

$$s' = \psi_2(v') = 1.5$$

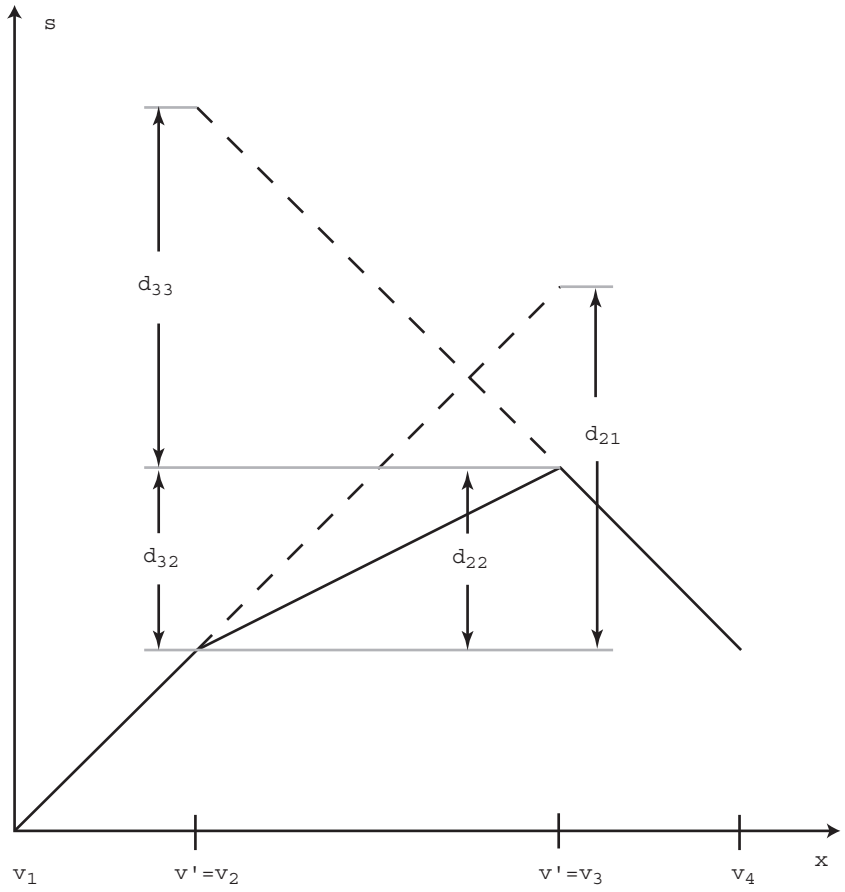

Figure 3.4: Contractions based on field quadric error with subset placement

The error of contraction is the sum of the squares of the distances indicated in the figure, which is

$$e = \begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} \frac{5}{2} & -\frac{10}{2} \\ -\frac{10}{2} & \frac{25}{2} \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 2.5$$

How meaningful that error measure is needs justification.

3.2.3 Minimization of the Absolute Field Error

As mentioned earlier, if the simplex to be contracted is not on the boundary of the mesh, then we can leave the domain error out of consideration, and concentrate

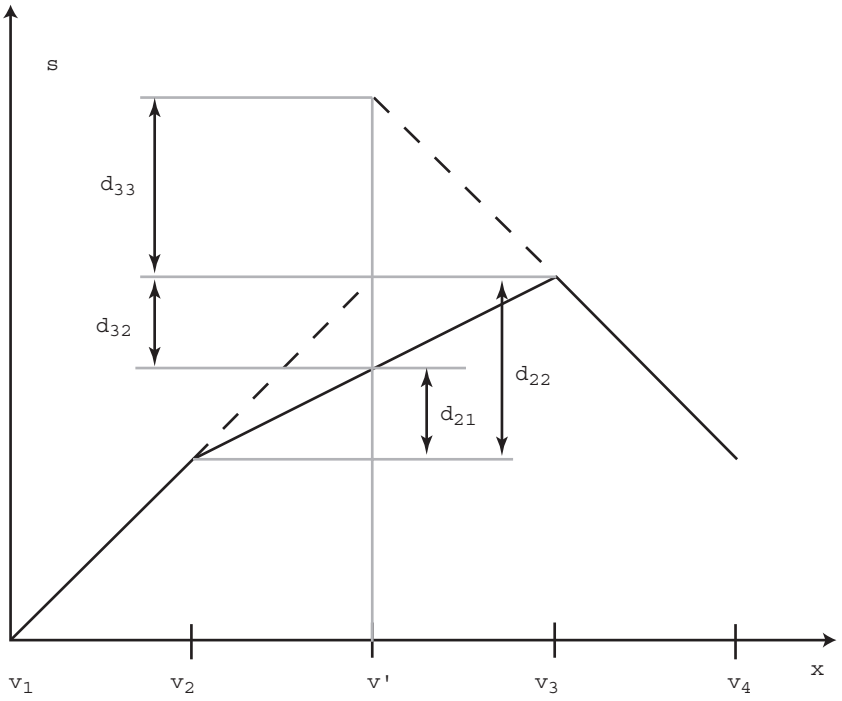

Figure 3.5: Contraction based on optimization of field quadric

only on the field error. A meaningful choice of the field error is

$$e(\psi_\Sigma, \psi_{\Sigma'}) = \max_{\mathbf{p}\in\Omega} \, |\, \psi_\Sigma(\mathbf{p}) - \psi_{\Sigma'}(\mathbf{p}) \,| \tag{3.1}$$

This is a true error metric in the sense that if Σ is first contracted to Σ' and Σ' is then contracted to Σ'', then

$$e(\psi_\Sigma, \psi_{\Sigma''}) \leq e(\psi_\Sigma, \psi_{\Sigma'}) + e(\psi_{\Sigma'}, \psi_{\Sigma''})$$

Thus, we obtain an upper bound on the total error due to the resulting multi step simplification by accumulating the errors in successive contractions.

The error measure in (3.1) involves a continuum of points in the domain of the mesh, and therefore, it is impractical to evaluate let alone to be considered for optimization. However, since ψ_Σ and $\psi_{\Sigma'}$ are piecewise linear over their respective meshes, the maximum difference between the two occurs at one of the vertices of the polyhedra formed by intersection of Σ and Σ'. That is,

$$e(\psi_\Sigma, \psi_{\Sigma'}) = \max_{\mathbf{p}\in\mathcal{V}_p} \, |\, \psi_\Sigma(\mathbf{p}) - \psi_{\Sigma'}(\mathbf{p}) \,| \tag{3.2}$$

where $\mathcal{V}_p$ is the set of vertices at which edges of Σ intersect faces of Σ' and vice versa. $\mathcal{V}_p$ is illustrated in Figure 2.4 for a 2D mesh.

In one dimension (3.2) further reduces to

$$e(\psi_\Sigma, \psi_{\Sigma'}) = e(v', s') = \max_{x \in \mathcal{V} \cup \mathcal{V}'} | \psi_\Sigma(x) - \psi_{\Sigma'}(x) |$$

Referring to the illustrative example, the error expression above becomes

$$\begin{aligned} e(v', s') &= \max \{ | \psi_1'(v_2) - s_2 | , | \psi_2'(v_3) - s_3 | , | \psi_2(v') - s' | \} \\ &= \max \{ (1 - \frac{s'}{v'}), (2 - 3\,\frac{1 - s'}{4 - v'} - \frac{4s' - v'}{4 - v'}), (s' - \frac{1}{2}\,v' - \frac{1}{2}) \} \end{aligned}$$

which is the maximum absolute difference before and after contraction between the fields at v_2 and v_3 (the endpoints of σ_2) and at v' (the final vertex of contraction). Values of v' and s' that minimize $e(v', s')$ can be found by solving

$$1 - \frac{s'}{v'} = 2 - 3\,\frac{1 - s'}{4 - v'} - \frac{4s' - v'}{4 - v'} = s' - \frac{1}{2}\,v' - \frac{1}{2}$$

as

$$v' = \sqrt{7} = 2.65$$

and

$$s' = \frac{7 + 2\sqrt{7}}{6} = 2.05$$

The corresponding mesh is shown in Figure 3.6.

In higher dimensions, minimization of the error expression in (3.2) requires calculation of the points in $\mathcal{V}_p$ and the errors at these points. However, the points of $\mathcal{V}_p$ are determined by the choice of $\mathbf{v}'$, and the errors at these points by s'. Clearly, the only way to solve the problem is a brute force search approach, which is computationally intractable. In the following section, we will present a heuristic argument to obtain a suboptimal solution.

3.3 Scalar Value Assignment Problem

As mentioned in the previous subsection, minimization of the maximum absolute field error in (3.1) is a computationally difficult problem for surface or volume

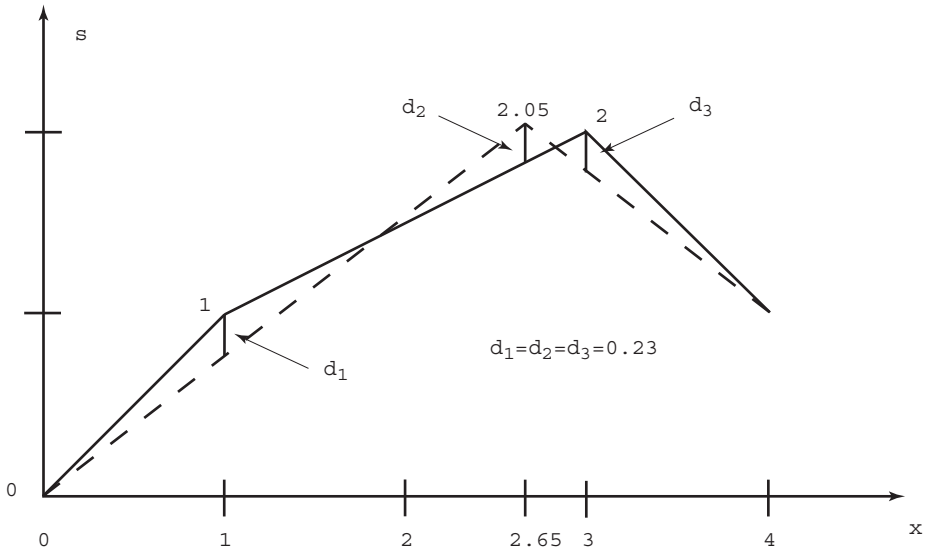

Figure 3.6: Contraction based on maximum absolute field error

meshes. A partial simplification can be reached by restricting $\mathbf{v}'$ to a few candidates and thus transforming the problem to a one-dimensional minimization over s'. That $\mathbf{v}'$ which results in the minimum error among the candidates can then be selected as the final vertex position. However, even when $\mathbf{v}'$ is fixed, minimization of the maximum field error requires calculation of the errors at the points of $\mathcal{V}_p$, which depend on s' assigned to $\mathbf{v}'$. Instead of searching for the best value of s' we propose to settle down with a reasonable suboptimal choice.

Assume that $\mathbf{v}' \in \tau$ is fixed (at one of the candidates of the final vertex position). Let $\sigma_i, i = 1, \ldots, k$, denote the neighbors of τ including τ itself, and define

$$\begin{aligned} s'_M &= \max_{1 \le i \le k} \{ \psi_i(\mathbf{v}') \} \\ s'_m &= \min_{1 \le i \le k} \{ \psi_i(\mathbf{v}') \} \\ s' &= \frac{s'_M + s'_m}{2} \end{aligned} \tag{3.3}$$

where ψ_i are the linear interpolants associated with σ_i. (Note that in the case of a volume mesh, if τ is a tetrahedron, then σ_i includes all vertex, edge and face neighbors of τ.)

Let ψ'_M, ψ'_m and ψ' denote the piecewise linear fields associated with the simplified mesh (after contraction of τ to $\mathbf{v}'$) corresponding to the assignments $\psi'(\mathbf{v}') = s'_M$, $\psi'(\mathbf{v}') = s'_m$ and $\psi'(\mathbf{v}') = s'$ to $\mathbf{v}'$, respectively. We now claim that for any $\mathbf{p} \in \Omega$,

$$|\psi'(\mathbf{p}) - \psi(\mathbf{p})| \leq \frac{s'_M - s'_m}{2} \tag{3.4}$$

so that

$$e_p = \frac{s'_M - s'_m}{2} \tag{3.5}$$

is an upper bound of the field error due to contraction of τ to $\mathbf{v}'$.

To prove the claim it suffices to consider only the points in the polyhedron

$$\mathcal{P}_\tau = \bigcup_{i=1}^{k} \sigma_i$$

consisting of τ and its neighborhood as $\psi'(\mathbf{p}) = \psi(\mathbf{p})$ for all $\mathbf{p} \in \Omega - \mathcal{P}_\tau$. Contraction of τ to $\mathbf{v}'$ eliminates τ and its edge and face neighbors, resulting in a tetrahedrization of $\mathcal{P}_\tau$ into $\sigma'_j, j = 1, \ldots, k'$, such that one of the vertices of each σ'_j is $\mathbf{v}'$ and the other three are boundary vertices of $\mathcal{P}_\tau$.

Consider an arbitrary point $\mathbf{p} \in \mathcal{P}_\tau$, which belongs to one of σ'_j, and therefore, can be expressed as a convex combination of its vertices as

$$\mathbf{p} = c'\mathbf{v}' + \sum_{j=1}^{3} c_j \mathbf{v}_{ij} \tag{3.6}$$

where $\mathbf{v}_{ij}$ are the vertices of σ'_j that are on the boundary of $\mathcal{P}_\tau$, and $0 \leq c', c_j \leq 1$. On noticing that

$$\psi'_M(\mathbf{v}_{ij}) = \psi'_m(\mathbf{v}_{ij}) = \psi'(\mathbf{v}_{ij}) = s_{ij}$$

for the boundary vertices, (3.6) implies that

$$\psi'_M(\mathbf{p}) = c's'_M + \sum_{j=1}^{3} c_j s_{ij} \geq c'\psi(\mathbf{v}') + \sum_{j=1}^{3} c_j \psi(\mathbf{v}_{ij}) = \psi(\mathbf{p})$$

$$\psi'_m(\mathbf{p}) = c's'_m + \sum_{j=1}^{3} c_j s_{ij} \leq c'\psi(\mathbf{v}') + \sum_{j=1}^{3} c_j \psi(\mathbf{v}_{ij}) = \psi(\mathbf{p})$$

that is,

$$\psi'_m(\mathbf{p}) \leq \psi(\mathbf{p}) \leq \psi'_M(\mathbf{p})$$

(3.6) also implies that

$$\begin{aligned}\psi'(\mathbf{p}) &= c's' + \sum_{j=1}^{3} c_j s_{ij} \\ &= \frac{1}{2}\left(c's'_M + \sum_{j=1}^{3} c_j s_{ij}\right) + \frac{1}{2}\left(c's'_m + \sum_{j=1}^{3} c_j s_{ij}\right) \\ &= \frac{\psi'_M(\mathbf{p}) + \psi'_m(\mathbf{p})}{2}\end{aligned}$$

From the last two relations we obtain (see Figure 3.7)

$$|\psi'(\mathbf{p}) - \psi(\mathbf{p})| \leq \frac{\psi'_M(\mathbf{p}) - \psi'_m(\mathbf{p})}{2} = \frac{c'(s'_M - s'_m)}{2} = c'e_p \leq e_p$$

proving the claim.

We now illustrate our heuristic on the 1D mesh of Figure 3.1. We consider three choices of the final vertex position, $v' = 1.5$, $v' = 2.0$, and $v' = 2.5$.

For $v' = 1.5$,

$$\psi_1(v') = 1.5, \psi_2(v') = 1.2 = s'_m, \psi_3(v') = 3.5 = s'_M$$

and therefore, the value assigned to v' and the corresponding predicted error are

$$s' = \frac{3.5 + 1.2}{2} = 2.35 \quad \text{and} \quad e_p = \frac{3.5 - 1.2}{2} = 1.15$$

For $v' = 2.0$,

$$\psi_1(v') = 2.0, \psi_2(v') = 1.5 = s'_m, \psi_3(v') = 3 = s'_M$$

and

$$s' = \frac{3.0 + 1.5}{2} = 2.25 \quad \text{and} \quad e_p = \frac{3.0 - 1.5}{2} = 0.75$$

Finally, for $v' = 2.5$,

$$\psi_1(v') = \psi_3(v') = 2.5 = s'_M, \psi_2(v') = 1.75 = s'_m,$$

and

$$s' = \frac{2.5 + 1.75}{2} = 2.125 \quad \text{and} \quad e_p = \frac{2.5 - 1.75}{2} = 0.375$$

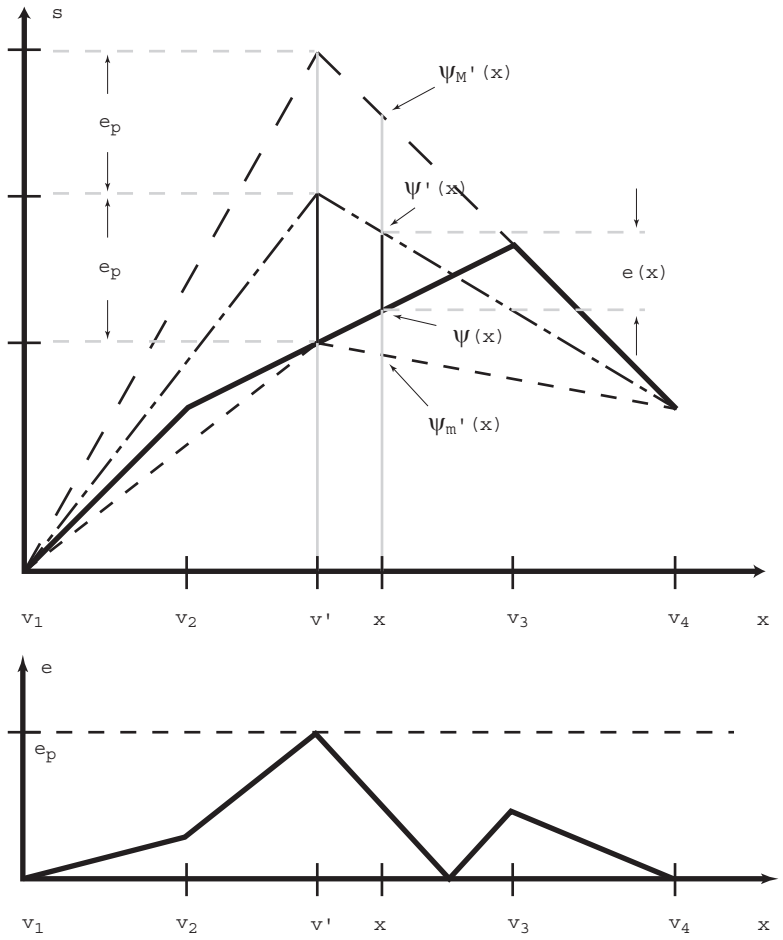

Figure 3.7: Relative magnitudes of ψ'_M, ψ'_m,ψ' and ψ at $\mathbf{p}$

The contractions based on the three choices of v' above and the scalar values assigned to them according to (3.3) are shown in Figure 3.8. Incidentally, the pair $(v', s') = (2.5, 2.125)$ turns out to be the best choice among all possibilities.

It should be observed that once the final vertex position $\mathbf{v}'$ is decided, our heuristic method not only provides a way of assigning the scalar value of the final vertex, but also gives an estimate of the field error due to the contraction. Besides, it is a general method that can be applied to meshes of arbitrary dimension as long as the linear interpolants ψ_i of the individual cell are known, and it does not differentiate whether the simplex to be contracted is an edge, a face or a higher order one. In this study, we implement simplification of a tetrahedral mesh by

means of contraction of tetrahedra and edges of tetrahedra. Clearly, contraction of faces, which is an in-between case, can also be implemented easily.

To conclude our discussion on the scalar value assignment problem, we present in Table 3.1 a comparison of the maximum absolute field errors due to the contractions by the methods considered in the previous section and by our heuristic approach. We observe that, for the illustrative example considered, our heuristic method is definitely better than the field error quadric approach of Cignoni et al, and is quite close to the generalized quadric of Garland and Heckbert.

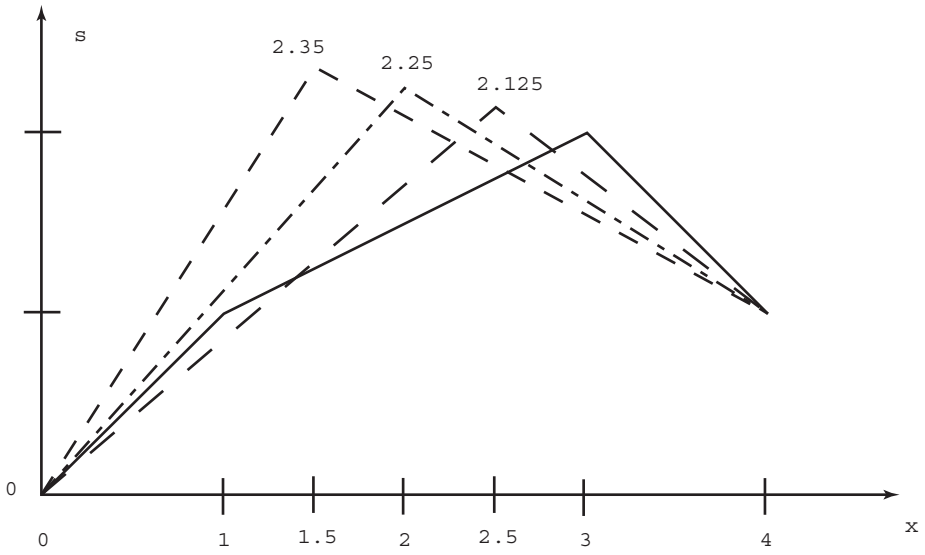

Figure 3.8: Contractions based on heuristics

Table 3.1: A comparison of the error of various contractions for the illustrative example

G-H	Cignoni	SVA	Absolute
0.2500	0.7500	0.3750	0.2257

Chapter 4

Implementations Details and Examples

In this chapter, we present the implementation details of our heuristic scalar value assignment (SVA) method introduced in the previous chapter. In the implementation, we considered iterative simplification of a volumetric dataset (with a tetrahedral mesh) by means of both tetrahedron and edge contractions. The SVA method is explained for tetrahedron contraction; however, implementation of the method for edge contraction is very similar. The small differences between the two implementations and problems unique to edge contraction approach are described in a separate section. By these two implementations, it is believed that the performance of a possible surface collapse method can be predicted. The performances of tetrahedron and edge contraction approaches using SVA are compared using two examples.

4.1 Data Structures

The volume data to be simplified are stored in four structures: A *vertex array*, a *data array*, a *tetrahedron array* and a *connectivity array*. Structures of these arrays and their relations are shown in Figure 4.1.

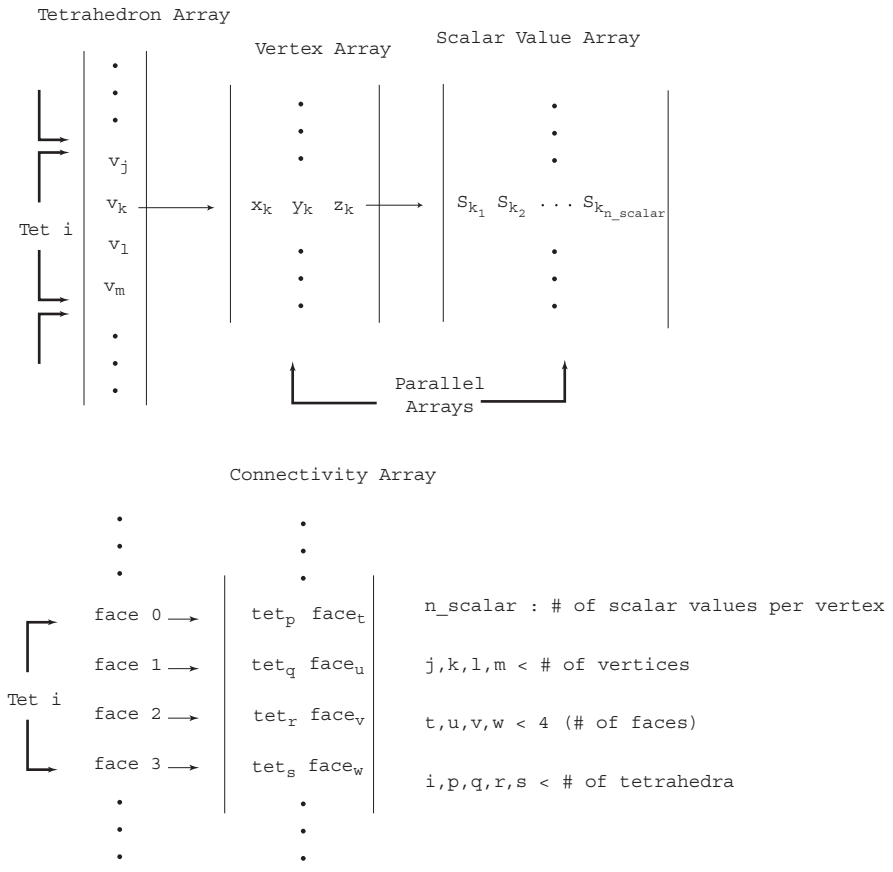

Figure 4.1: Data structures

The vertex array is an $n \times 3$ dimensional array that stores for each vertex $\mathbf{v}_i$ the triple x_i, y_i, z_i defining the position of $\mathbf{v}_i$, where n is the number of vertices of the original mesh.

The data array stores for each vertex $\mathbf{v}_i$ a set of n_s scalar values each representing the sample values of a different scalar field. Simplification is performed on the basis of one of these scalar values.

The $m \times 4$ tetrahedron array keeps for each tetrahedron σ_j in the mesh the indices of the four vertices of σ_j, where m is the number of tetrahedra.

The final structure is an $m \times 8$ array that keeps the connectivity information, which is defined over the faces of the tetrahedra. For every tetrahedron σ_j there

are four entries corresponding to the faces of σ_j and each consisting of two parts: The index of the neighboring tetrahedron σ_k, and the face index of σ_k from which σ_k is connected to σ_j.

In addition to the basic structures above are a couple of work arrays. One is a *priority queue* in which information about contractable tetrahedra in the current mesh is stored. The priority queue is implemented as a binary heap in this thesis. Each heap element consists of three entries: The tetrahedron index, the predicted error to be introduced if that tetrahedron is contracted, and the final vertex position of contraction. The error is also used as the key value of the heap.

A second work array is an *error array*, which stores for each vertex $\mathbf{v}_i$ a scalar error value e_i. Initially, all vertices are assigned zero error. When a tetrahedron is contracted, three of its four vertices are removed from the vertex array and the fourth one is replaced with the final vertex of contraction. The predicted error associated with the contracted tetrahedron is then assigned to the final vertex, and the vertices that are removed are marked by assigning a sentinel value to their errors. The error of a vertex is later added to the overall predicted error if a tetrahedron including this vertex is collapsed.

The power of incremental simplification methods comes from the locality of the operations, which allows for working with a small portion of the dataset at a time. In our study, locality is facilitated by means of two *neighbors arrays*, which store the indices of the neighbors of a *tetrahedron in process* (TIP).

4.2 Tetrahedron Contraction

The general algorithm used in this implementation is shown in Table 4.2. It consists of three stages: Initialization, simplification, and finalization. In the initialization part, a pass over the whole tetrahedron array is executed in order to find all contractAble tetrahedra and insert them into the heap together with their final vertex positions and predicted errors. In the simplification part, tetrahedra

in the heap are contracted one at a time, and the data and the heap are updated accordingly. In the finalization part, the data arrays are cleaned of removed tetrahedra and vertices. Next, the basic steps of the algorithm are explained.

Figure 4.2: Pseudocode for tetrahedron collapse

```
        /* Initial pass */
for every tetrahedron σ do
   find neighbors of σ
   determine if σ is contractable
   compute the predicted error, scalar value and position of the final vertex
   insert σ into priority queue

        /* Iterative simplification */
while error is within bound
   extract the tetrahedron σ with minimum error
   find neighbors of σ
   contract σ
   determine the effected tetrahedra
   for every effected tetrahedron τ do
      find neighbors of τ
      compute the predicted error, scalar value and position of the final vertex
      update the position of τ in the priority queue

        /* Finalize */
write the simplified volume data to disk
```

4.2.1 Determination of the neighbors

Various steps of the algorithm require neighbors of a given tetrahedron σ. A function determines the neighbors of σ, and returns their indices in two arrays, one containing the indices of the face and edge neighbors of σ, and the other

indices of the vertex neighbors. Although edge and vertex neighbor information is not explicitly held, edge or vertex neighbors are face neighbors to other vertex, edge or face neighbors. Eventually, any neighbor is a face neighbor of a neighbor and therefore, it is possible to reach any edge or vertex neighbor. This observation allows for finding all neighbors of σ by a breadth-first-search (BFS) method over the connectivity array.

The BFS is initiated by inserting σ into a queue. The first element of the queue is extracted as the TIP. If a face neighbor τ of the TIP is not previously checked (colored) and if it shares at least one vertex with σ, then it is inserted into the queue and is colored. If τ shares a single vertex with σ, it is included into vertex-neighbors array together with the index of the shared vertex. Otherwise, it is included into the face/edge-neighbors array. When the queue is empty, the function returns the neighbors arrays.

A naive method for finding the neighbors would be to search the entire tetrahedron array. The running time of such an algorithm would take $\Theta(m)$. Theoretically the running time of the BFS method is also $\mathcal{O}(m)$. However, in practice, a tetrahedron has much less neighbors that require an average running time much less than the naive approach. Since neighbors of the same tetrahedron are needed more than once at various steps of the algorithm, such a saving in the time complexity becomes extremely important.

4.2.2 Contractability check and boundary preservation

For every tetrahedron σ in the current mesh a number of candidates for the final vertex position (FVP) are determined depending on whether σ has a primitive (a vertex, and edge or a face) on the boundary of the mesh.

In the easier case when σ has no primitive on the boundary, choice of the candidates for the FVP is arbitrary. In this study five candidates, the four vertices of σ and their midpoint, are considered, although it is possible to consider additional candidates at the expense of increasing computation time (Midpoints

of the edges and faces of σ, and/or the midpoints of the corner subtetrahedra formed by a vertex and the midpoints of the edges incident on that vertex, are reasonable choices of the additional candidates for the FVP). Then, σ is checked for contractability to each candidate FVP $\mathbf{v}'$, and if it is found to be contractable to $\mathbf{v}'$ the corresponding predicted error is calculated. If σ is contractable to at least one candidate FVP, it is inserted into the heap together with the FVP that results in the smallest predicted error.

The test for contractability of σ to $\mathbf{v}'$ involves determining whether the contraction operation results in an inconsistency in the mesh, which occurs when a vertex neighbor of σ occupies part of the region occupied by another neighbor. Such an inconsistency arises when the shared vertex of a neighbor flips sides with respect to the plane formed by the other three vertices after contraction. Flipping of a vertex neighbor is illustrated in Figure 4.3 for a 2D mesh.

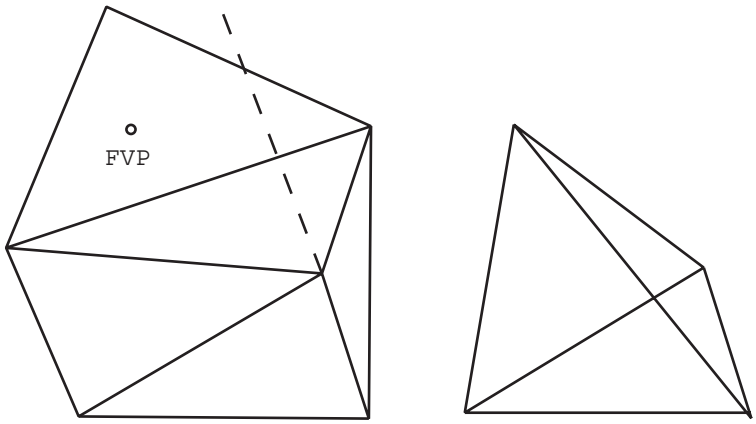

Figure 4.3: Flipping of a triangle in a 2D mesh

The test whether a vertex neighbor τ of σ flips involves a couple of triple vector products. Let τ have the vertices $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_4$, with $\mathbf{v}_0$ being the vertex common to τ and σ. The cross product of the vectors $\mathbf{w}_1 = \mathbf{v}_1 - \mathbf{v}_3$ and $\mathbf{w}_2 = \mathbf{v}_2 - \mathbf{v}_3$

$$\mathbf{n} = \mathbf{w}_1 \times \mathbf{w}_2$$

points in the direction of the normal of the plane formed by $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$. The sign of the inner product of $\mathbf{n}$ with the vector $\mathbf{w}_0 = \mathbf{v}_0 - \mathbf{v}_3$ and $\mathbf{w}' = \mathbf{v}' - \mathbf{v}_3$ indicate whether these vectors point in the same direction $\mathbf{n}$. As a result, a flipping occurs if and only if

$$(\mathbf{n}^T\mathbf{w}_0)(\mathbf{n}^T\mathbf{w}') < 0$$

When a tetrahedron σ has a primitive on the boundary, requirement to preserve the boundary after the contraction imposes additional constraints on the FVP. We consider several possibilities separately.

If σ has primitives on two disjoint surfaces (e.g., the top and bottom surfaces of a cube), contraction of σ results in a deformation of at least one of these surfaces. If it has more than one face on the boundary, then contraction results in a deformation along the edge or at the vertex these faces meet. In these two cases, σ is not contracted.

If σ has a single vertex or a single edge on the boundary, then the boundary can be preserved by choosing the FVP at the vertex or along the edge on the boundary. In such a case σ is contractable to the chosen FVP provided no flipping of a vertex neighbor occur.

If σ has a single face α (a triangle) on the boundary, then the problem becomes one of surface simplification. In our implementation, the normal of α is compared to the normal of each of its neighbors (triangles on the boundary). If all the angles between the normal of α and the normals of its neighbors are smaller than a predetermined interval, then σ can be contracted to a point on α, provided such a choice of the FVP does not result in a flipping. In this case, contraction of σ introduces a small domain error.

4.2.3 Error prediction

Let, as usual, σ be a tetrahedron to be contracted to FVP $\mathbf{v}'$, and let σ_i be the neighbors of σ, $i = 1, \ldots, k$. As explained in the previous chapter, to find the

value s' to be assigned to $\mathbf{v}'$ and calculate the corresponding predicted error, we need to evaluate each of the linear interpolants ψ_i at $\mathbf{v}'$. As we already considered in Section 2.3,

$$\psi_i(\mathbf{v}') = \mathbf{s}_i^T \mathbf{c}_i = c_{i0}s_{i0} + c_{i1}s_{i1} + c_{i2}s_{i2} + c_{i3}s_{i3}$$

where s_{ij}'s are the scalar values associated with the vertices of σ_i, and c_{ij}'s are obtained by solving the 4×4 system

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x_{i0} & x_{i1} & x_{i2} & x_{i3} \\ y_{i0} & y_{i1} & y_{i2} & y_{i3} \\ z_{i0} & z_{i1} & z_{i2} & z_{i3} \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_{i0} \\ c_{i1} \\ c_{i2} \\ c_{i3} \end{bmatrix} \tag{4.1}$$

In our implementation, instead of the 4×4 system in (4.1), we solve the equivalent 3×3 system

$$\begin{bmatrix} x' - x_{i3} \\ y' - y_{i3} \\ z' - z_{i3} \end{bmatrix} = \begin{bmatrix} x_{i0} - x_{i3} & x_{i1} - x_{i3} & x_{i2} - x_{i3} \\ y_{i0} - y_{i3} & y_{i1} - y_{i3} & y_{i2} - y_{i3} \\ z_{i0} - z_{i3} & z_{i1} - z_{i3} & z_{i2} - z_{i3} \end{bmatrix} \begin{bmatrix} c_{i0} \\ c_{i1} \\ c_{i2} \end{bmatrix} \tag{4.2}$$

which is obtained from (4.1) on noting that

$$c_{i0} + c_{i1} + c_{i2} + c_{i3} = 1 \tag{4.3}$$

Once c_{i0}, c_{i1} and c_{i2} are found from (4.2), c_{i3} can be calculated from (4.3) easily.

Replacing the 4×4 system in (4.1) by the equivalent 3×3 system in (4.2) saves quite significant computing time.

4.2.4 Contraction of a tetrahedron

When a tetrahedron σ is contracted to a final vertex $\mathbf{v}'$

- σ disappears,
- all face and edge neighbors of σ also disappear,

- vertex neighbors of σ are modified,
- mesh topology changes, and
- some tetrahedra which were previously incontractable may become contractable, and vice versa.

When σ is contracted it is not actually erased from the tetrahedron array; instead, a flag associated with σ is turned on to indicate that it does not exist anymore. The same is done for all the face and edge neighbors of σ, whose indices are kept in the face/edge neighbors array.

Similarly, three of the vertices of σ are flagged in the vertex array to indicate that they do not exist in the simplified mesh. The location of the fourth vertex is used to store the position (x', y', z') of $\mathbf{v}'$. At the same time, the index of the shared vertex of any vertex neighbor of σ is changed to the index of this fourth vertex. This takes care of modification of the vertex neighbors of σ.

Removal of σ and its neighbors change the mesh topology, i.e., the connectivity information. Obviously, the vertex neighbors are only modified and do not cause a change in the topology directly. Since the connectivity information is defined over the faces of tetrahedra, removal of a face neighbor of σ does not change the topology either, because a face neighbor reduces to an edge after contraction of σ. Now, consider an edge neighbor τ of σ. When σ is contracted, τ collapses to a triangle on which two of its faces (with a common base and tips connected with the shared edge) merge together as shown in Figure 4.4. Then the two face neighbors of τ become face neighbors themselves sharing this triangle. In this case the neighbor information of the two face neighbors of τ are modified to point at each other and this ensures that no tetrahedron in the current mesh points to τ which is practically removed from the mesh.

Finally, since all vertex neighbors of σ are modified when σ is contracted, Their contractability properties and the associated predicted errors change. A vertex neighbor, which is previously incontractable may be contractable after its modification, in which case it is included into the heap. Even if it was in the

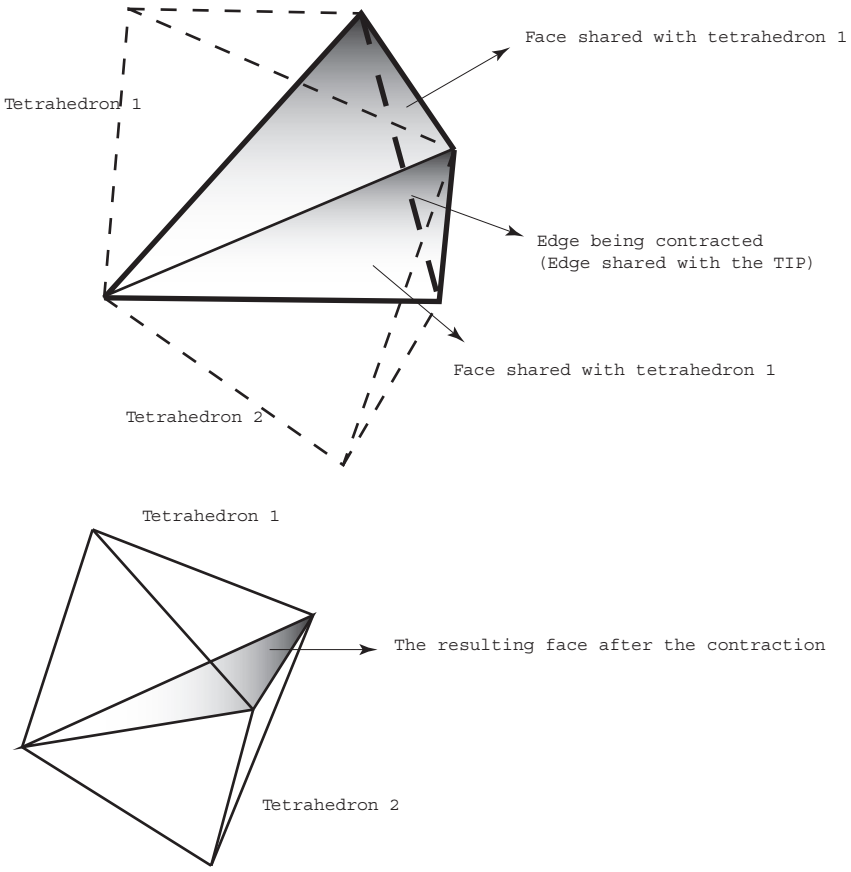

Figure 4.4: Modification of the mesh after removal of an edge neighbor

heap prior to contraction of σ and remains contractable after the contraction operation, its associated predicted error changes, necessitating a modification in its position in the heap. Conversely, a vertex neighbor of σ which is previously contractable may be incontractable, in which case it must be removed from the heap. Unfortunately, vertex neighbors of σ are not the only tetrahedra which needs o be checked for contractability. Their neighbors are also effected by the contraction of σ and must be tested again for contractability. The set of all tetrahedra that will be retested are called the *effected tetrahedra*. In order to determine the effected tetrahedra, a similar method to the one used in finding the neighbors of a tetrahedron is used. The only difference is that a tetrahedron is checked if it shares a vertex with any of the modified tetrahedra.

4.2.5 Data cleaning

Once the program exits the main loop, the mesh will contain redundant information. This is due to the fact that the removal process is just a marking instead of actually altering the data structures. Therefore, a final step in the algorithm extracts the useful information, vertices and tetrahedra still existing in the current mesh, and arranges these arrays. A naive way of doing it would be scanning the tetrahedron array for an existing tetrahedron and placing it in the first available position in the array. Of course all the connectivity information of must change their neighbor indices at every step. The running time of such an algorithm would be $\mathcal{O}(n^2)$. However, the use of a map can reduce the running time to $\mathcal{O}(n)$. A pass over the tetrahedron array is used to construct a map. The map array keeps the new location for a tetrahedron and therefore is of size n. Then, by using the information in the map array, the tetrahedron array can be modified in a single pass.

4.3 Edge contraction

Although edge contraction is very similar to tetrahedron contraction there are some implementational problems associated with edge contraction. In the general sense, the data structure is based on tetrahedra; therefore, a number of steps must be taken for the edge contraction method to work properly and efficiently.

One method of dealing with the data structure would be to extract the implicit edge information from the current dataset (by numbering edges and associated vertices, and by determining the neighborhood relations). Although this is manageable, we preferred the use the current dataset to have the option of switching between tetrahedron and edge collapse methods whenver desired.

Every tetrahedron consists of exactly six edges, and these edges can be numbered just like its faces. Therefore, in order to specify a specific edge, the index of the tetrahedron and the edge number are sufficient. Edges are likely to be shared

by many tetrahedra and can be numbered in as many different ways as the number of tetrahedra sharing the edge. This gives rise to the problem of inserting multiple copies of the edge into the heap while processing different tetrahedra. Checking whether an edge had already been inserted into heap is very time consuming; instead, an ordering over the tetrahedra does solve the problem. In the tetrahedron contraction method every tetrahedron keeps an index to its position in the heap. However, in the edge contraction method, a tetrahdron needs to keep six indices for its edges. A sentinel value indicates that the edge is not in the heap either because it is not contractable or it is already in the heap with a different tetrahedron index.

In the initialization stage all tetrahedra are processed sequentially, and all the edges of a tetrahedron are processed before the next tetrahedron is processed. When a tetrahedron is processed it is colored. Consider a tetrahedron σ and a face neighbor τ, where σ is processed before τ. τ will not process the edges shared by σ (a face neighbor shares three edges) because they were already processed. In this way, an edge is only processed once and there are no multiple entries into the heap.

One other situation where the ordering is important is when dealing with the effected edges, which must be processed after the contraction of an edge. As in the tetrahedron contraction method, all the effected tetrahedra are found and their colors are reset. In order to process an edge only once (as in the initialization stage), the tetrahedra which are effected but not modified are processed first and the modified tetrahedra are processed later.

The rest of the of the methods in edge contraction are similar to the ones used in tetrahedron contraction with only trivial modifications.

4.4 Examples

The SVA method was experimented with two datasets.

The first dataset is a rather small one with cube-shaped domain. It is created by *MeshGenerator*, which was implemented as part of this study. The MeshGenerator divides a hexagonal cell into six tetrahedra such that two hexagonal cells can be put side by side or one on top of the other without causing incompatibilities with the tetrahedra. In the example, we generated a mesh of $20 \times 20 \times 20$ unit cubes, having a total number of $m = 48.000$ tetrahedra. The scalar field of the dataset was chosen (quite arbitrarily) as

$$\psi(x, y, z) = xyz$$

The second dataset is the famous blunt dataset taken from NASA. The dataset consists of five different scalar fields; however, we used only the pressure field to run our experiments. The blunt data is the output of a finite volume simulation run over a blunt fin at mach 2. The pressure field displays a good view of the shock wave produced on the leading edge of the fin and bends over the fin.

In both of the examples considered, the error was expressed in percent. For this purpose, the difference between the maximum and minimum values of the scalar field values over the dataset is defined as the *scalar value range*, and the absolute error was scaled by this value.

The results of our simplification method using SVA are tabulated in Tables 4.1 and 4.2 for different values of the predicted error. In both tables "Max error" corresponds to the case where no more contraction is possible. The following symbols are used in the tables.

- n : Number of vertices (n_0: # of vertices in the original mesh)
- m : Number of tetrahedra (m_0: # of tetrahedra in the original mesh)
- N : Number of iterations to simplify the mesh until the error hits the indicated percent
- T : Simplification time in sec

The experiments were carried out on a PentiumIV 1.5 GHz CPU with 256 MBytes of RD-RAM.

In simplification of the cube, tetrahedron and edge contractions resulted in comparable cell counts for a given error bound; although edge contraction, working with smaller units, was expected to provide a better simplification. The reason is perhaps the regularity of the dataset, which was made up of identical unit cubes. The more realistic blunt data, however, confirmed our expectation, resulting in a reduction of the cell count to about 20% of the original data for 5% error bound in the case of edge contraction as opposed to about 53% in the case of tetrahedron contraction.

Another observation is that edge contraction requires many more iterations than tetrahedron contraction for a given error bound. This is a completely expected result because a typical mesh contains many more edges than tetrahedra. What is interesting, however, is that number of iterations per second in tetrahedron and edge contractions are comparable. In fact, contraction of a tetrahedron takes a little less time than the contraction of an edge. Although an edge has relatively less number of neighbors with respect to a tetrahedron, the greater heap size might cause such an outcome.

The original and simplified datasets are visualized by using Koyamada's direct volume visualization technique. As an aid for debugging, a code which renders only the boundary of a mesh, was developed. This is an interactive application in which the camera can fly through and around the volume. This enables the user to see whether the boundary was preserved or not. It can also be used to visualize the vertices before and after simplification.

Finally, the images of the original and simplified datasets for both examples are given in Figures 4.5, 4.6, 4.7 and 4.8.

Table 4.1: Experimental Results: Cube Dataset

Dataset	:	Cube
		$n_0 = 9261$
		$m_0 = 48000$

Method	:	Tetrahedron contraction				
Initialization Time	:	6.80 sec				
% error	n	m	m/m_0	N	T	N/T
0.1	8,679	44,673	93.07	194	9.06	21.4
0.2	5,427	26,425	55.05	1,278	54.59	23.4
0.3	4,359	20,943	43.63	1,634	64.54	25.3
0.5	3,492	16,352	34.07	1,923	73.19	26.3
1.0	2,400	10,577	22.04	2,287	82.68	27.7
5.0	1,518	5,850	12.19	2,581	91.09	28.3
Max	1,326	4,740	9.88	2,645	93.56	28.3

Method	:	Edge contraction				
Initialization Time	:	9.80 sec				
% error	n	m	m/m_0	N	T	N/T
0.1	7,026	37,011	77.11	2,235	107.34	20.8
0.2	5,413	29,788	62.06	3,848	181.91	21.2
0.3	4,740	26,282	54.75	4,521	212.84	21.2
0.5	3,976	21,788	45.39	5,285	249.39	21.2
1.0	3,067	16,331	34.02	6,194	293.89	21.1
5.0	1,678	7,864	16.38	7,583	363.74	20.7
Max	940	3,315	6.91	8,321	401.67	20.7

n :	# of vertices
m :	# of tetrahedra
N :	# of iterations
T :	Time in seconds

Table 4.2: Experimental Results: Blunt's Dataset

Dataset	:	Blunt
		$n_0 = 40960$
		$m_0 = 187395$

Method	:	Tetrahedron contraction				
Initialization Time	:	35.09 sec				
% error	n	m	m/m_0	N	T	N/T
0.1	40,438	184,723	98.57	174	12.25	14.2
0.2	39,142	178,058	95.02	606	42.86	14.1
0.3	37,570	169,784	90.60	1,130	80.31	14.1
0.5	34,180	152,014	81.12	2,260	156.14	14.5
1.0	29,428	127,381	67.97	3,844	243.42	15.8
5.0	24,082	99,107	52.87	5,826	330.80	17.6
Max	22,735	91,745	48.96	6,075	350.22	17.2

Method	:	Edge contraction				
Initialization Time	:	44.16 sec				
% error	n	m	m/m_0	N	T	N/T
0.1	28,047	138,775	74.05	12,913	968.34	13.3
0.2	23,689	120,837	64.48	17,271	1,245.22	13.9
0.3	20,953	108,942	58.13	20,007	1,410.50	14.2
0.5	17,720	94,091	50.21	23,240	1,596.79	14.6
1.0	13,604	74,093	39.54	27,356	1,823.24	15.0
5.0	7,017	37,837	20.19	33,943	2,178.38	15.6
Max	2,640	11,279	6.02	38,320	2,437.33	15.7

n :	# of vertices
m :	# of tetrahedra
N :	# of iterations
T :	Time in seconds

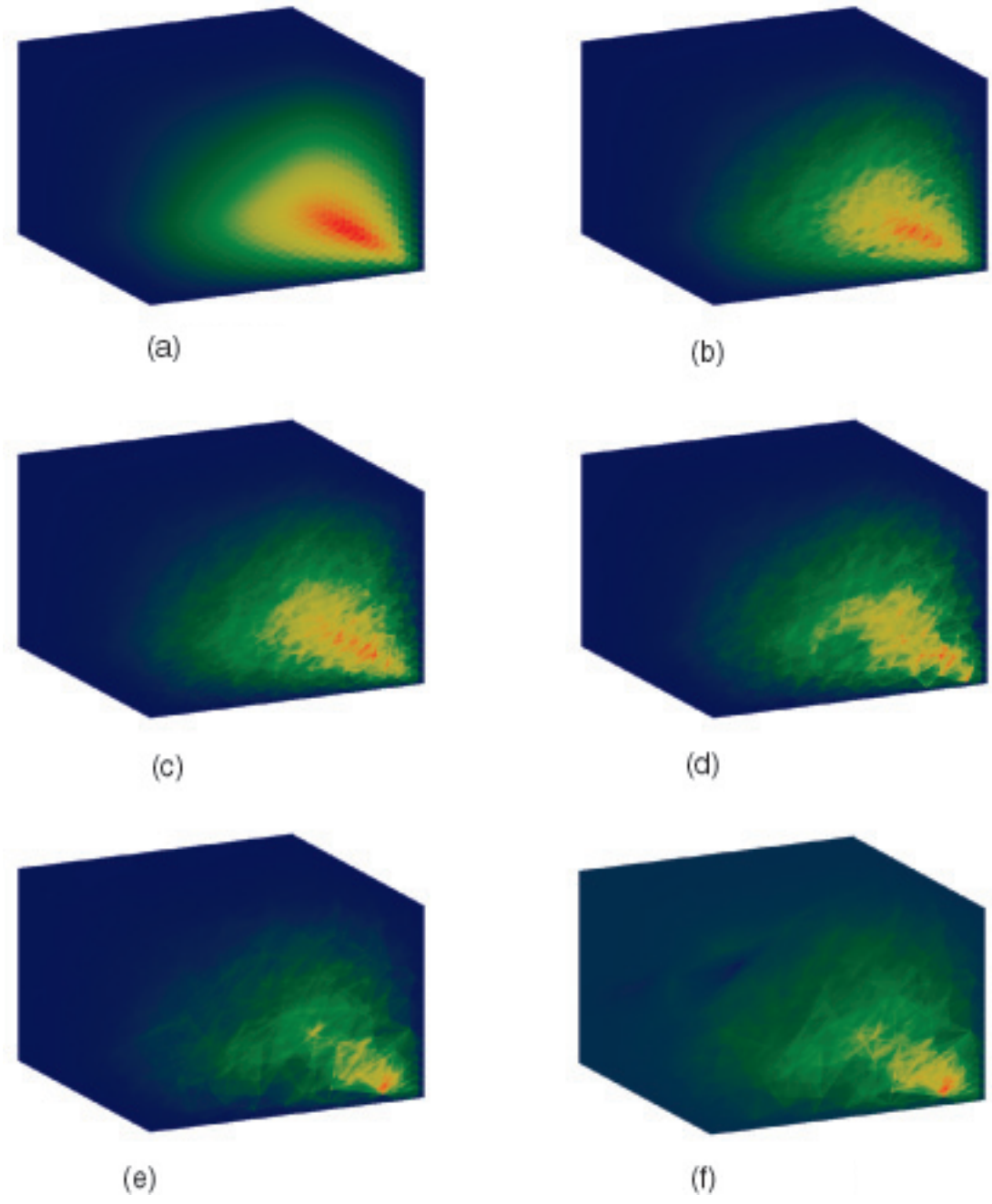

Figure 4.5: Images of cube data with tetrahedron collapse. (a) Original. (b) 0.2 % error. (c) 0.5 % error. (d) 1.0 % error. (e) 5.0 % error. (f) Max error

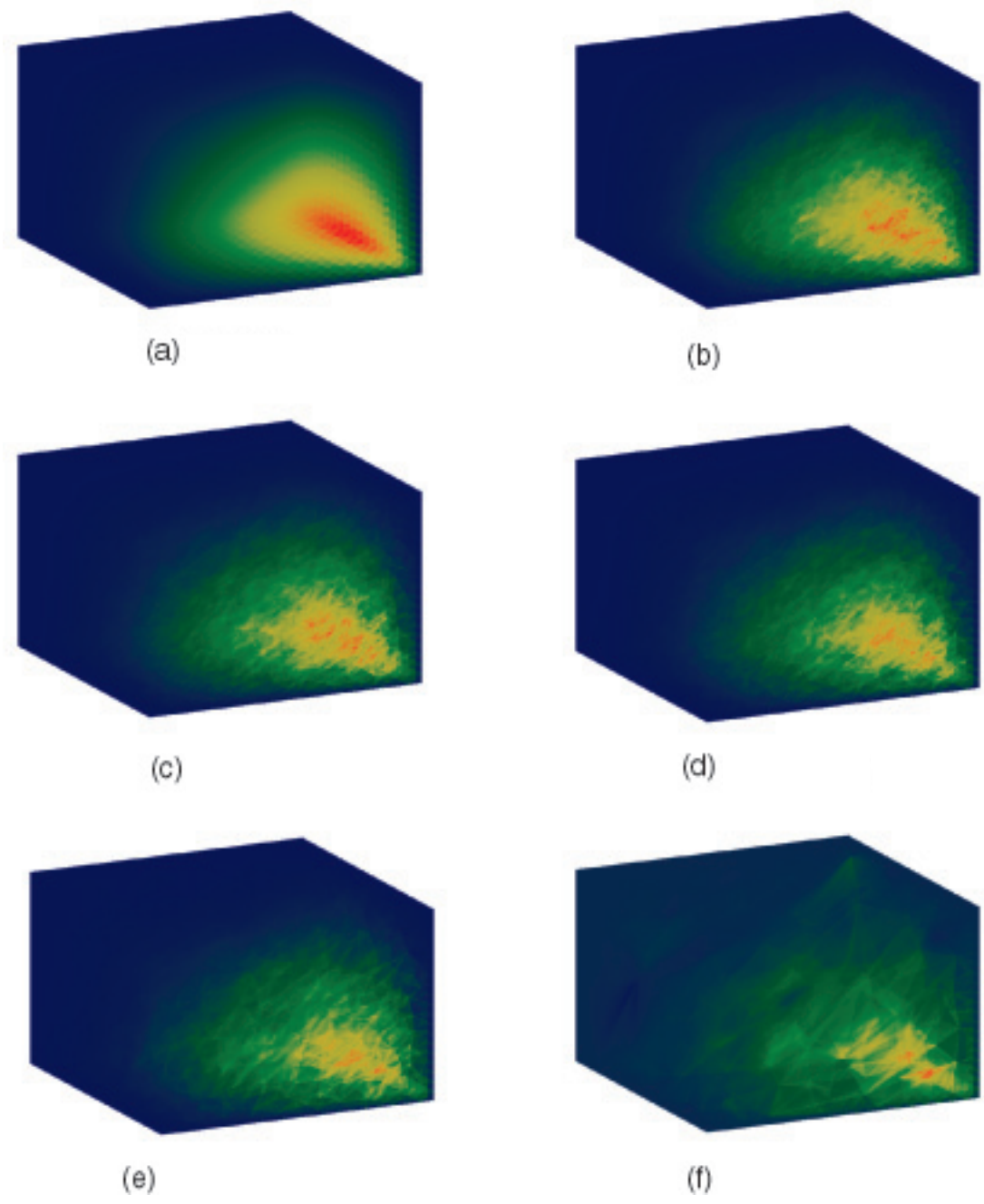

Figure 4.6: Images of cube data with edge collapse. (a) Original. (b) 0.2 % error. (c) 0.5 % error. (d) 1.0 % error. (e) 5.0 % error. (f) Max error

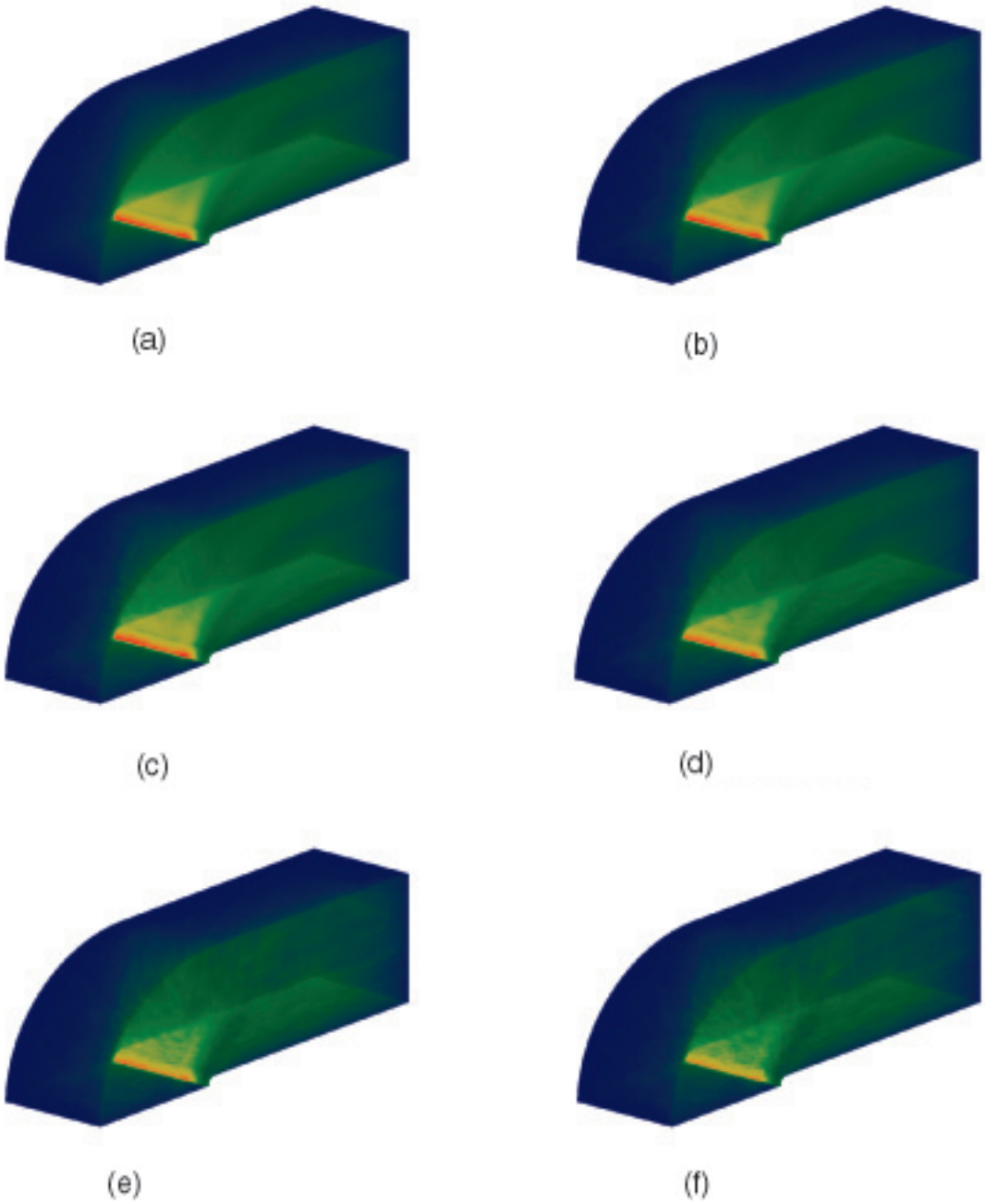

Figure 4.7: Images of blunt data with tetrahedron collapse. (a) Original. (b) 0.2 % error. (c) 0.5 % error. (d) 1.0 % error. (e) 5.0 % error. (f) Max error

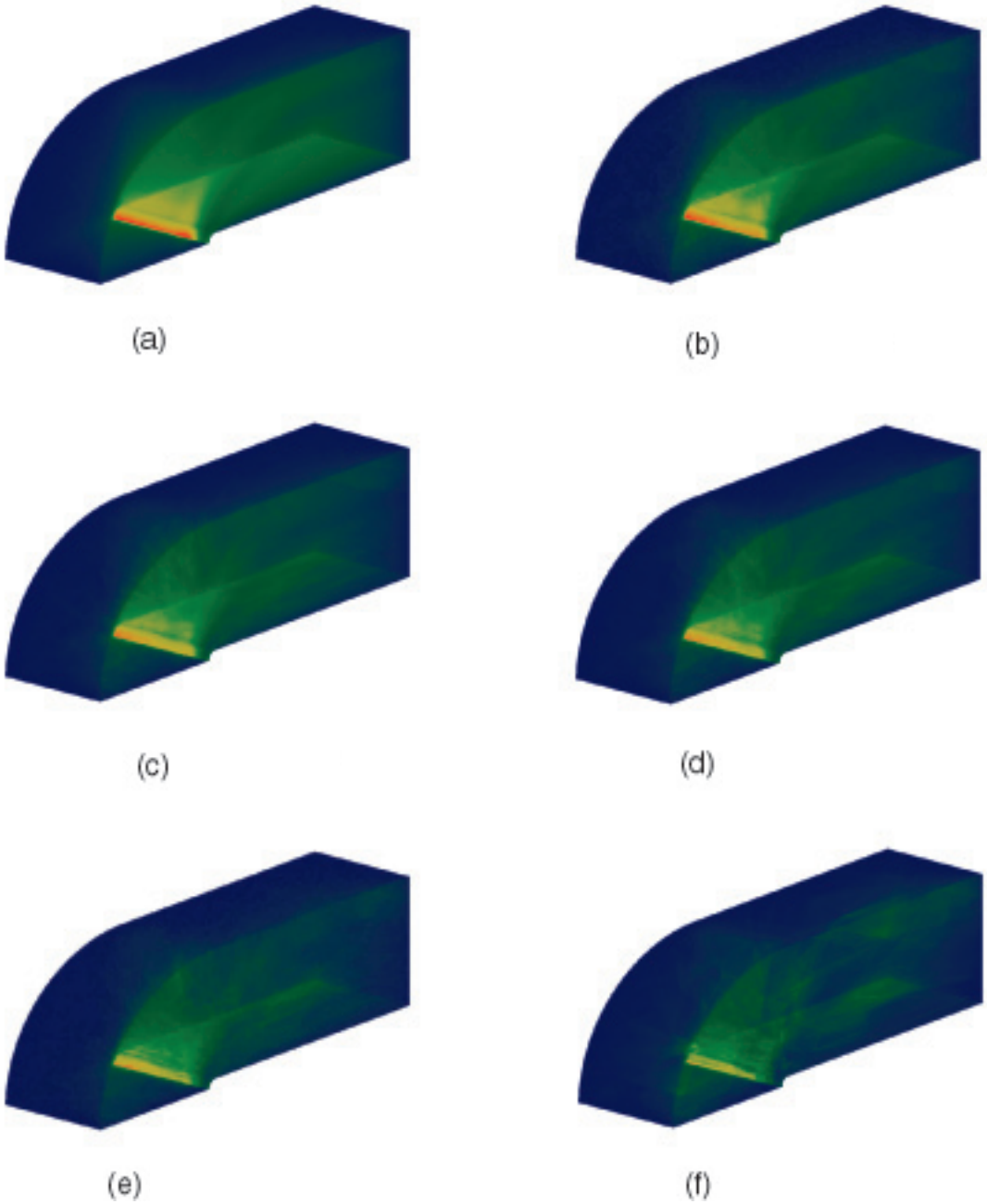

Figure 4.8: Images of blunt data with edge collapse. (a) Original. (b) 0.2 % error. (c) 0.5 % error. (d) 1.0 % error. (e) 5.0 % error. (f) Max error

Chapter 5

Conclusions and Future Work

In this thesis, we proposed two simplification methods for volumetric datasets represented as unstructured tetrahedral grids. The simplification algorithms use tetrahedron contraction and edge contraction based on scalar value assignment.

The results obtained by the SVA method shows that the heuristic works quite well. The method is easy to implement, fast and effective. Tetrahedron contraction method works much faster than the edge contraction method, however, the edge contraction method is a finer method and can do more drastic simplification than the tetrahedron collapse method.

Tetrahedron contraction can be viewed as a combination of three sequential edge contractions that reduce the tetrahedron to a vertex. Therefore, the chance of a tetrahedron being contractible is lower with respect to an edge. This explains why tetrahedron contraction algorithm can not do drastic simplification.

Two things are worth some more effort in this thesis. First, the edge contraction method can be modified in order to increase its performance. One approach could be extracting the edge information implicit in the current dataset before the main algorithm. Second, a hybrid method can be applied in which the algorithm starts with contracting tetrahedra, then switching to face or edge contractions. As mentioned earlier, it is possible to contract any degree of simplicial in the mesh.

The face contraction method was not implemented because the performance can be predicted from the two extreme ends of tetrahedron and edge contraction.

In order for a hybrid method to work, the heap must be able to store different types of primitives (edges, faces and tetrahedra). This can be done by using a union structure in C. However, the contraction part would have to be extended to deal with all three types. It is expected that a hybrid method will have better performance and will be a finer method than tetrahedron contraction.

There are also a couple of modifications that can be done to the implementations in general. First of all, the heap elements store unnecessary information. This is due to the fact that the initialization stage is run once and the heap is stored for later use. This enables the user to run the initialization phase only once and get multiple records of simplification without the initialization. A better approach would be to store tetrahedra related information in another array and store it also along with the heap.

One other improvement can be done while checking for contractability. In our implementations, for every candidate final vertex position (FVP), every vertex neighbor is checked for flipping. Checking for flipping on a neighbor involves computation of the 3×3 α matrix of the tetrahedron. Instead of doing this for every candidate, it is more convenient to calculate the α matrix of a vertex neighbor and check for flipping with every candidate FVP (the number of candidates is much less than the number of vertex neighbors). In the latter case, the function is likely to return before all the vertex neighbors are checked and therefore improve the average running time.

The calculation of the predicted error for every candidate FVP is again a similar problem described in the contractability check. In order to determine the error and scalar value at FVP, the scalar value at the FVP must be calculated with respect to every neighbor. In the current implementations, every candidate FVP is taken into account and all the scalar fields are calculated. However, in this case the α matrices of all the neighbors are calculated as many times as the number of candidate FVPs.

In general the SVA and error prediction heuristic works quite well and the performance can be improved further by some small modifications and possibly with the implementation of a hybrid method.

Bibliography

[1] P. Cignoni, C. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. *IEEE Visualization' 2000.*

[2] P. Cignoni, L. De Floriani, C. Montoni, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. In A. Kaufman and W. Krueger, editors, *1994 Symposium on Volume Visualization*, pages 19–26, 1994.

[3] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, /1997.

[4] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. *ACM Computer Graphics*, 30(Proc. of SIGGRAPH '96):119–128, 1996.

[5] C. Forest, H. Delingette, and N. Ayache. Removing tetrahedra from a manifold mesh. In *Proc. IEEE Computer Animation, 2002.*

[6] M. Garland. Multiresolution modeling: Survey and future opportunities. In *Eurographics '99 – State of the Art Reports*, pages 111–131, 1999.

[7] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *ACM Computer Graphics*, 31(Proc. of SIGGRAPH '97):209–216, 1997.

[8] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *IEEE Visualization '98*, pages 263–270, 1998.

[9] M. Garrity. Raytracing irregular volume data. *Proc. of 1990 WS on Volume Visualization*, 24(5):35–40.

[10] B. Hamann and J. Chen. Data point selection for piecewise trilinear approximation. *Computer Aided Geometric Design*, 11:477–489, 1994.

[11] H. Hoppe. Progressive meshes. *ACM Computer Graphics*, 30(Proc. of SIGGRAPH '96):99–108, 1996.

[12] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geometric Design*, 8:123–142, 1991.

[13] K.-L. Low and T.-S. Tan. Model simplification using vertex clustering. In *Proc. 1997 Symp. Interactive 3D Graphics*, pages 75–82, 1997.

[14] D. Luebke. A survey of polygonal simplification algorithms. Technical Report TR97-045, 16, 1997.

[15] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. *ACM Computer Graphics*, 31(Proc. of SIGGRAPH '97).

[16] E. Puppo and R. Scopigno. Simplification, LOD and Multiresolution - Principles and Applications, 1997.

[17] K. J. Renze and J. H. Oliver. Generalized unstructured decimation. *IEEE Computer Graphics & Applications*, 16(6):24–32, 1996.

[18] J. Rossignac and P. Borrel. Multi-resolution 3D approximation for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Geometric Modeling in Computer Graphics*, pages 455–465, 1993.

[19] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *ACM Computer Graphics*, 26(Proc. of SIGGRAPH '97).

[20] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *ACM Computer Graphics*, 26(Proc. of SIGGRAPH '96):65–70, Nov. 1990.

[21] I. J. Trotts, B. Hamann, K. I. Joy, and D. F. Wiley. Simplification of tetrahedral meshes. *IEEE Visualization*, pages 287–295, 1998.

[22] P. Williams. Interactive splatting of nonrectilinear volumes. *Visualization '92 Proceedings*, pages 37–45, 1992.