David F. Wiley
June 2003
Department of Computer Science

Approximation and Visualization of Scientific Data Using Higher-order Elements

Abstract

The trend in science and engineering applications has been to produce larger data sets, since computers and imaging technology are getting faster and storage space is increasing. Large amounts of data are difficult to visualize, and it is impossible to directly visualize them on inexpensive computers. Many visualization techniques exist that visualize certain types of large data. However, a general solution does not exist. A hierarchical method provides the foundation for a solution. Linear and quadratic decomposition elements can be used to form an approximation hierarchy representing large data; a user can then visualize this hierarchy on low-end machines. A hierarchical approximation method is described that uses linear, quadratic, and curved-quadratic decompositional elements. Linear element approximation and visualization has been studied extensively in the past. Higher-order element approximation and visualization is not nearly as developed as that for linear elements, thus, more research is needed. Fundamental visualization techniques—such as isosurfacing, ray casting, and cutting planes—for quadratic elements are described.

Approximation and Visualization of Scientific Data Using Higher-order Elements

By

DAVID F. WILEY
B.S. (University of California, Davis) 1998
M.S. (University of California, Davis) 2001

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Department of Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Committee in charge

2003

Approximation and Visualization of Scientific Data Using Higher-order Elements

To my parents, Joanne and Frederick Wiley, and my family.

Acknowledgments

I would like to give special thanks to my advisor, Bernd Hamann. His careful crafting of explanations and drawing of diagrams, so that I could understand complicated concepts, has allowed me to learn very much during my time of working with him. He is a model researcher and teacher. I appreciate the help received from all associated with the Center for Image Processing and Integrated Computing, including Ken Joy, Nelson Max, Kwan-Liu Ma, Martin Bertram, Falko Küster, Jörg Meyer, Lars Linsen, Gunther Weber, Ben Gregorski, and Hank Childs. I would like to acknowledge my parents for their persistence in expecting my best and never making college an option for me. I know that my mom would be very proud of me. I would like to thank all of my friends that supported me after the time of her passing. They made it possible for me to finish. Especially, I would like to thank my friends Dan Harris, Bryan Kalbfus, Erika Scott, Colin Johnson, Robert Chute, Sapna Bhuta, Karim Mahrous, Janine Bennett, and Haeyoung Ha. I also thank Mark Oser for steering me in the direction of computer programming. Lastly, I would like to recognize my siblings, Pam, Steve, Chris, Tina, and Karen, and their support during my life, and also the support and encouragement from the rest of my family. Thanks.

Contents

List of Figures

Abstract

The trend in science and engineering applications has been to produce larger data sets, since computers and imaging technology are getting faster and storage space is increasing. Large amounts of data are difficult to visualize, and it is impossible to directly visualize them on inexpensive computers. Many visualization techniques exist that visualize certain types of large data. However, a general solution does not exist. A hierarchical method provides the foundation for a solution. Linear and quadratic decomposition elements can be used to form an approximation hierarchy representing large data; a user can then visualize this hierarchy on low-end machines. A hierarchical approximation method is described that uses linear, quadratic, and curved-quadratic decompositional elements. Linear element approximation and visualization has been studied extensively in the past. Higher-order element approximation and visualization is not nearly as developed as that for linear elements, thus, more research is needed. Fundamental visualization techniques—such as isosurfacing, ray casting, and cutting planes—for quadratic elements are described.

Chapter 1

Introduction

1.1 Motivation

Computing power and storage has been growing at phenomenal rates during the past 50 years. Scientists are taking advantage of these improvements by developing more complex simulations and by recording more information. Growth in complexity and information leads to an increase in the data set sizes generated by scientists. While the information growth provides more accurate and higher-resolution data to researchers, these large data sets are difficult to view. Often, the supercomputer that generated the data cannot be used for visualization of the data, since its time is reserved for other tasks; and visualization on low-end machines is difficult when the data is several times larger than the core memory. This is the reason for the large-scale visualization problem that the scientific community is facing today.

Analysis of the problem reveals the guidelines that bound it. First, the ultimate goal is to produce a "meaningful" image of the data. The allowable time to create this image depends upon the context. It is generally acceptable to a user to see poor representations of the data when previewing, positioning, navigating, and exploring the data. However, when the user finds something interesting, a higher-quality image is made to show a truer representation of the data. Second, for a very-large data set, there are far more data points in the data set then there are pixels on a computer screen. Consider the question of whether all of the data is needed for visualization; since it is impossible—on low-end machines—to

visualize all of the data in a reasonable amount of time, a data reduction step must take place. Third, in many scientific applications, it is very important for the scientist to know precisely how accurate the representation image is. Following these guidelines leads to a solution.

Approximation of data is needed, since the amount of data must be reduced. Reliable error estimates—so that users can gauge the quality of an approximation—require the approximation technique to be mathematically sound. Since varying quality images are required for previewing and close inspection there must also be corresponding approximations to draw from. This leads to a two-step solution: a pre-processing step that approximates the data, possibly yielding many approximations of varying quality, and a visualization step that renders the approximations.

A natural solution to this problem is to represent the data as a hierarchy of increasingly better approximations. An appropriate hierarchy consists of a top level that coarsely describes the data set with few data points. (This top level generally has a high error associated with it.) Subsequent levels increase quality by using more data points. Once constructed, selecting and visualizing an appropriate level in the hierarchy finds a solution to the large-scale visualization problem.

1.2 Research Goals of Dissertation

The main research goal is to promote the use of higher-order elements. Visualization of higher-order elements assumes that there are higher-order elements to visualize, thus, methods to create meaningful higher-order element data sets are discussed. A linear element approximation method is described as a foundation for a higher-order element approximation method.

In general, it is important to have a hierarchical data-dependent approximation scheme to solve the large data visualization problems. A simplex-based approach is powerful, since simplex-based schemes are more general and can be used to decompose complex domains—i.e., using intervals, triangles, and tetrahedra for the 1D, 2D, and 3D cases, respectively, as opposed to arbitrary polyhedra. Two approximation methods using these

attributes are discussed here (both are refinement methods). The first method uses linear simplices as the approximation element. On a high level, these steps describe the approach:

1. Begin with a coarse decomposition of the domain—the convex hull of all data sites (the term the "triangulation" generally refers to the decompositional mesh).
2. Compute an approximation over the triangulation.
3. Compute error estimates for each element (interval, triangle, or tetrahedron).
4. Bisect the element having the highest error.
5. Repeat steps 2, 3, and 4 until the estimated global approximation error is under a user-specified tolerance.

The second method uses a quadratic simplex instead of a linear simplex as the approximation element. A single quadratic element has the advantage of representing a complex region where many linear elements might be required to approximate that same region. (The hierarchical construction is the same as in the linear case.) Consider having two hierarchical approximations—one is linear and the other is quadratic—that are stored across a network from the inexpensive machine that is used to visualize the hierarchy (each hierarchy, level-for-level, approximates the data to the same error tolerance). The quadratic hierarchy is transmitted more quickly (than the linear hierarchy) across the network, since fewer elements are required per level. (This scenario is analogous to transmitting elements across the video bus inside a computer.)

Linear simplices are well studied in the visualization community, and graphics hardware exists that can process millions of linear triangles per second. Thus, rendering of linear-triangle hierarchies is relatively easy. The community has also extensively researched volumetric visualization based on space decompositions of linear tetrahedral elements, thus, many visualization techniques exists to handle these types of elements.

Research opportunities exist in the visualization of higher-order elements. Little work has been done in regard to visualizing them. Tessellating a higher-order element with smaller linear elements and visualizing the linear elements is the standard method for visualization. This works, but the question of whether there are more efficient means still exists.

Methods to directly visualize higher-order elements are needed. Fundamental methods for visualization, including isosurfacing, ray casting, and cutting planes, are described here for quadratically defined elements. In many cases, the ideas developed in this dissertation can be extended to elements having higher order than quadratic.

Part I

Approximation

One goal of data approximation is to represent data using a smaller, more concise, and more manageable form. There are many types of data that can benefit from being approximated, a few examples are shown in Figure 1.1. In each of these examples, there are data points that do not *need* to be included in a representation of the data. These data points often introduce redundant information that can be represented accurately even if it were approximated.

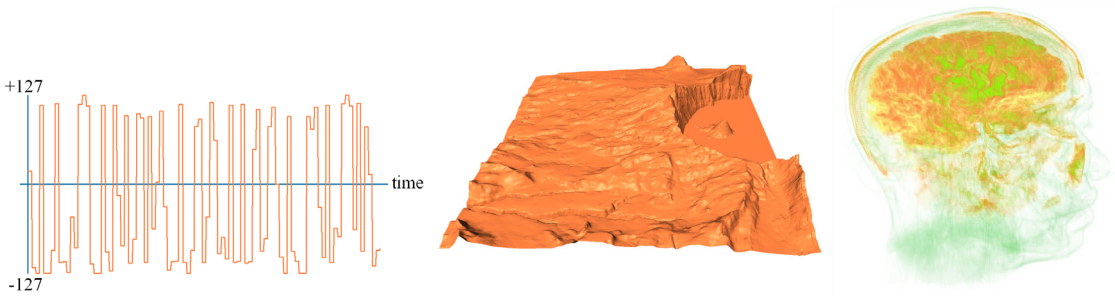

Figure 1.1: Examples of data that can benefit from approximation. Left image shows a digitized sound wave. Middle image shows a height field. Right image shows an MRI of a human head.

A method for the construction of hierarchies of single-valued functions in one, two, and three dimensions is described. The input to the method is a coarse decomposition of the compact domain of a function in the form of an interval (1D case), triangles (2D case), and tetrahedra (3D case). Best linear spline approximations are computed, understood in an integral least squares sense, for functions defined over such triangulations and are refined using repeated bisection. This requires the identification of the simplex with largest error and splitting it into two simplices. Each bisection step requires the re-computation of all spline coefficients due to the global nature of the best-approximation problem. Nevertheless, this can be done efficiently by bisecting multiple simplices in one step and by using an optimized sparse matrix solver.

Different methods are known and used for the hierarchical representation of very large data sets. Unfortunately, only a small number of these methods are based on a well developed mathematical theory. In the context of visualizing very large data sets in 2D and 3D, it is imperative to develop hierarchical data representations that allow us to visualize and analyze physical phenomena at various levels of detail. General, robust, and efficient methodologies are needed to support the generation of hierarchical data representations and

their applicability for the visualization process.

The construction of hierarchies of triangulations and best linear spline approximations of functions are described here. The main idea underlying the construction of a data hierarchy is repeated bisection of intervals, triangles, and tetrahedra. The coefficients associated with each vertex in a triangulation are computed in a best-approximation sense. (The terms "triangulation" or "mesh" are used to describe the general decomposition.) Whenever an element is bisected, due to a large local error, one needs to re-compute linear spline coefficients for all vertices in the new, refined triangulation.

1.3 Previous Work

There are many hierarchical methods targeted at approximating large data sets. For example, *wavelet* methods are described in [5, 15, 40]. The work described in [40] has the advantage of supporting both lossless and lossy compression. In general, wavelet methods work well for data lying on uniform, rectilinear, and power-of-two grids and provide fast and highly accurate compression.

Simplification methods using data elimination strategies are described in [7, 8, 16, 17, 20, 27, 28]. These methods provide a hierarchy by removing data points in a specific order to minimize the error at each level. Simplification concepts can be applied to both two- and three-dimensional data. These methods are more general than most wavelet methods, since arbitrary input meshes can be converted to a form appropriate for each method. Refinement methods similar to the ones discussed here are described in [19, 23, 39]. (Most data-dependent refinement methods—methods that consider the error relative to the data being approximated—can also be adapted to arbitrary meshes.)

Simplification methods begin with the highest resolution data mesh and remove data to produce the hierarchy. Another class of methods, *refinement* methods, begin with a coarse mesh—covering the domain of the data being approximated—and refine this mesh by inserting data points to produce the hierarchy, see [19, 39].

Data-dependent methods consider the data error and refine or simplify accordingly, as those previously mentioned. Non-data-dependent methods, such as the one discussed in

[13], are also being used. The method described in [11] performs an iterative "thinning step" based on radial basis functions on scattered points while maintaining a Delaunay triangulation. The method described in [11] is similar to [13] and suggests that data-dependent methods can better approximate input data by focusing data elements around high-gradient regions—regions where data values fluctuate produce more error (analytically and visually) when approximated with few elements.

Comparisons of wavelet, decimation, simplification, and data-dependent methods, including the methods discussed in [15, 20, 27, 39], are provided in [26]. This survey discusses the many approaches to surface simplification and also examines the complexity of some of the most commonly used methods.

Chapter 2

Linear Elements

Linear elements provide a foundation for the hierarchical data approximation method described here and demonstrate the capabilities of the system. Linearly defined simplices are used as the approximation elements that represent the function being approximated. An approximation consists of a set of approximation elements including line segments, triangles, and tetrahedra in the 1D, 2D, and 3D case, respectively.

2.1 The 1D Case

The goal is to represent a 1D (univariate) function $F(x)$ by a set of line segments (i.e., a piecewise-linear spline). A linear spline is defined by a set of N knots $k_i \in \mathbb{R},\ 0 \leq i \leq N-1$ and coefficients c_i positioned at these knot locations, forming a spline point $\mathbf{p}_i = (k_i, c_i)^{\mathrm{T}}$. The individual intervals are found by considering sequentially adjacent pairs of knots k_m and k_{m+1}, where $k_m < k_{m+1},\ 0 \leq m \leq N-2$. Figure 2.1 shows a linear spline representation of $F(x) = \sin(4\pi x^2),\ 0 \leq x \leq 1$.

Representing the linear spline approximation $A(x)$ as a linear combination $A(x) = \sum_{i=0}^{N-1} c_i f_i(x)$ of independent functions $f_0(x),\ \ldots,\ f_{N-1}(x)$, allows the use of the *best-approximation* method to find the coefficients c_i, see [10]. This is done by solving the

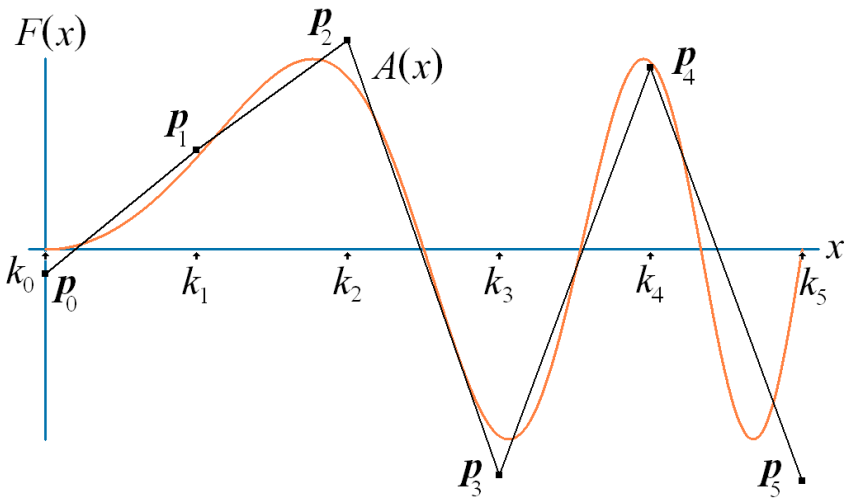

Figure 2.1: Linear spline representation of $F(x) = \sin(4\pi x^2),\ 0 \leq x \leq 1$. Orange curve shows original function $F(x)$. Black spline shows approximation $A(x)$. Knot locations k_i are indicated by small black arrows along x-axis (in this case, knots are distributed uniformly across the interval $[0, 1]$). Black squares denote spline points $\mathbf{p}_i$.

system

$$\begin{bmatrix} \langle f_0, f_0\rangle & \cdots & \langle f_{N-1}, f_0\rangle \\ \vdots & & \vdots \\ \langle f_0, f_{N-1}\rangle & \cdots & \langle f_{N-1}, f_{N-1}\rangle \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} \langle F, f_0\rangle \\ \vdots \\ \langle F, f_{N-1}\rangle \end{bmatrix}, \tag{2.1}$$

(abbreviated as $M\mathbf{c} = \mathbf{F}$) for $\mathbf{c}$, where f_i is an abbreviation for $f_i(x)$ and $\langle g, h\rangle$ is the scalar product of two functions $g(x)$ and $h(x)$, defined over the interval $[a, b]$ as

$$\langle g, h\rangle = \int_a^b g(x)h(x)dx. \tag{2.2}$$

The best-approximation method provides a globally optimal solution for the coefficients by minimizing the standard L_2 norm error metric for $D(x) = F(x) - A(x)$, the difference between the approximation $A(x)$ and the original function $F(x)$, defined as

$$\sqrt{\langle D, D\rangle} = \sqrt{\int_a^b \Big(F(x) - A(x)\Big)^2 dx}\ . \tag{2.3}$$

The system $M\mathbf{c} = \mathbf{F}$ is solved efficiently when using mutually orthogonal and normalized basis functions $f_i(x)$, i.e., $\langle f_i, f_j\rangle = \delta_{ij}$ (Kronecker delta). When using these basis functions, only the diagonal elements are non-zero, and the coefficients are given by $c_i = \langle F, f_i\rangle$. However, this does not guarantee a "good" approximation throughout an interval, since a basis function f_i only covers the knot location k_i; a basis function that covers the entire

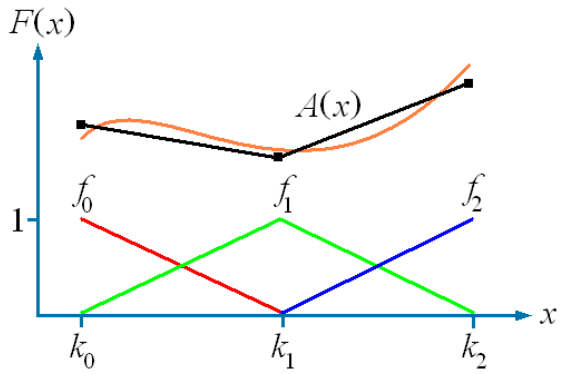

Figure 2.2: Best-approximation example. Orange curve shows the original function $F(x)$. Black line segments show the linear spline approximation $A(x)$. Basis functions f_0, f_1, and f_2 are shown for knot locations k_0, k_1, and k_2, respectively.

interval that it connects to provides a better approximation. One such basis function is the "hat" function. A hat function $f_i(x)$ takes on the values

$$f_i(x) = \begin{cases} 1, & \text{if } x = k_i; \\ \frac{k_{i+1}-x}{k_{i+1}-k_i}, & \text{if } k_i < x \leq k_{i+1}; \\ \frac{x-k_{i-1}}{k_i-k_{i-1}}, & \text{if } k_{i-1} \leq x < k_i; \\ 0, & \text{everywhere else.} \end{cases} \tag{2.4}$$

Figure 2.2 shows a linear spline approximation, knot locations, and associated hat basis functions. Using hat basis functions produces a tridiagonal linear system, since the only non-zero elements of M for row i are $\langle f_{i-1}, f_i \rangle$, $\langle f_i, f_i \rangle$, and $\langle f_{i+1}, f_i \rangle$. These scalar products are given by

$$\begin{aligned} \langle f_{i-1}, f_i \rangle &= \tfrac{1}{6}\Delta_{i-1}, \\ \langle f_i, f_i \rangle &= \tfrac{1}{3}\left(\Delta_{i-1} + \Delta_i\right), \text{ and} \\ \langle f_{i+1}, f_i \rangle &= \tfrac{1}{6}\Delta_i, \end{aligned} \tag{2.5}$$

where $\Delta_j = k_{j+1} - k_j$. Thus, M is the tridiagonal matrix

$$M = \frac{1}{6}\begin{bmatrix} 2\Delta_0 & \Delta_0 & & & & \\ \Delta_0 & 2(\Delta_0+\Delta_1) & \Delta_1 & & & \\ & \Delta_1 & 2(\Delta_1+\Delta_2) & \Delta_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & & \Delta_{N-2} & 2\Delta_{N-2} \end{bmatrix}. \tag{2.6}$$

Linear-time system solvers exist for tridiagonal linear systems. (For an arbitrary set of basis functions $f_i(x)$, one must investigate a means for an efficient solution to the linear system.)

The *global error* E for a linear spline approximation $A(x)$ is given by Equation (2.3). The *local error* e_i for interval i is defined as

$$e_i = \sqrt{\int_{k_i}^{k_{i+1}} \Big(F(x) - \big(c_i f_i(x) + c_{i+1} f_{i+1}(x)\big)\Big)^2 \, dx} \ , \ i = 0, \ \ldots, \ N-2. \qquad (2.7)$$

The scalar products $\langle F, f_i \rangle$ and the error values E and e_i are computed using numerical integration. Romberg integration is used to compute the required integrals, see [4, 25]. Romberg integration is discussed in the Appendix, see Section A.1.

2.2 The 2D Case

The goal is to represent a 2D (bivariate) function $F(x, y)$ by a set of triangles. It is assumed that the field (scalar function) to be approximated over a domain is known analytically. Should this not be the case, e.g., in the case of *scattered data* (when one is given a set of randomly distributed points with associated function values without connectivity information), it is possible to construct an analytical representation by performing a prior data interpolation or approximation step, see [13, 14, 36]. In the case that a data set is defined on a grid, the required analytical definition is given by a piecewise linear function for a simplicial (triangular) grid and a piecewise bilinear function in the case of quadrilateral grid cells. A linear spline is defined by a set of N_K knots $\mathbf{k}_i = (x_i, y_i)^{\mathrm{T}} \in \mathbb{R}^2, \ 0 \le i \le N_K - 1$, coefficients c_i positioned at these knot locations, and a set $\mathcal{T} = \{T_m\}$ of N_T knot triples $T_m = (\mathbf{k}_a^m, \mathbf{k}_b^m, \mathbf{k}_c^m), \ 0 \le m \le N_T - 1, \ 0 \le a, b, c \le N_K - 1, \ a \ne b \ne c$, defining the triangulation of the knots. Spline points $\mathbf{p}_i = (x_i, y_i, c_i)^{\mathrm{T}}$ are formed from the knot locations and coefficients. Figure 2.3 shows a linear spline representation of $F(x, y) = x^2 + y^2, \ -\frac{1}{2} \le x, y \le \frac{1}{2}$.

Representing the linear spline approximation $A(x, y)$ as a combination $A(x, y) = \sum_{i=0}^{N-1} c_i f_i(x, y)$ of independent functions $f_0(x, y), \ \ldots, \ f_{N-1}(x, y)$, allows the use of the best-approximation method to find the coefficients c_i, see Equation (2.1). The hat function $f_i(x, y)$ used in this case influences the *platelet* of triangles connected to knot $\mathbf{k}_i$, denoted as

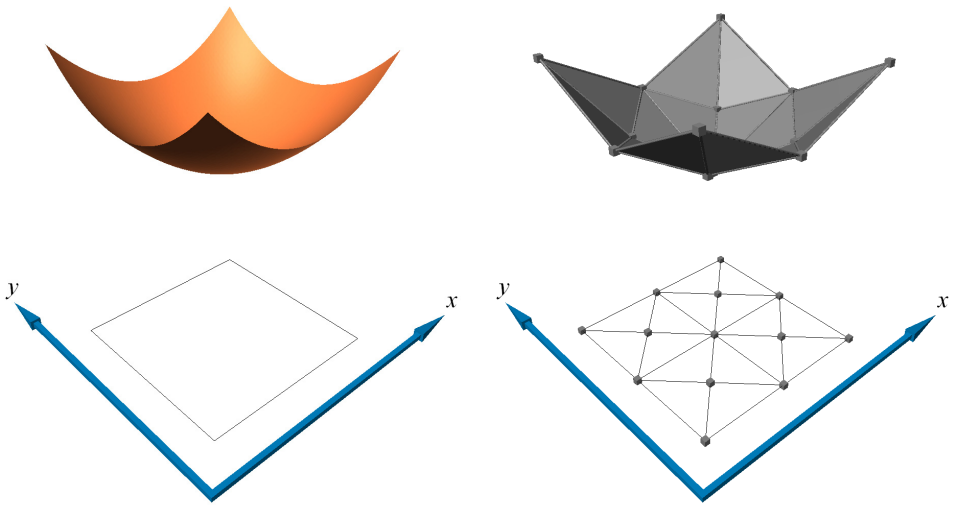

Figure 2.3: Linear spline representation of $F(x,y) = x^2 + y^2,\ -\frac{1}{2} \le x, y \le \frac{1}{2}$. Left image shows original function $F(x,y)$. Right image shows approximation $A(x,y)$. (Gray boxes in the xy-plane denote knot locations.)

$\mathcal{T}_i = \{T_j\},\ 0 \le j \le n_m - 1$, where n_m is the number of triangles in the platelet $\mathcal{T}_i$. Ordering the knots of T_j so that $\mathbf{k}_a^j = \mathbf{k}_i$, the basis function $f_i(x,y)$ takes on the values

$$f_i(x,y) = \begin{cases} 1, & \text{if } (x,y)^{\mathrm{T}} = \mathbf{k}_i; \\ 1 - u_j - v_j, & \text{if } (x,y)^{\mathrm{T}} \text{ is inside } T_j; \\ 0, & \text{everywhere else,} \end{cases} \tag{2.8}$$

where u_j and v_j for triangle T_j are found by solving

$$\begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{k}_b^j u_j + \mathbf{k}_c^j v_j + \mathbf{k}_i(1 - u_j - v_j) \tag{2.9}$$

for u_j and v_j, which are given as

$$u_j = \frac{(y_c^j - y_i)x + (x_i - x_c^j)y + x_c^j y_i - y_c^j x_i}{(y_b^j - y_c^j)x_i + (x_c^j - x_b^j)y_i + y_c^j x_b^j - x_c^j y_b^j} \tag{2.10}$$

and

$$v_j = \frac{(y_b^j - y_i)x + (x_i - x_b^j)y + x_b^j y_i - y_b^j x_i}{(y_b^j - y_c^j)x_i + (x_c^j - x_b^j)y_i + y_c^j x_b^j - x_c^j y_b^j}. \tag{2.11}$$

Figure 2.4 shows an example linear spline approximation, knot locations, and associated hat basis functions.

The 2D case requires integration over triangles. The *change-of-variables theorem* allows the integration over arbitrary triangles in $\mathbb{R}^2$ to the *standard triangle* $T^{\mathbb{U}}$ in parameter space $\mathbb{U}^2$ having vertices $(0,0)^{\mathrm{T}}$, $(1,0)^{\mathrm{T}}$, and $(0,1)^{\mathrm{T}}$.

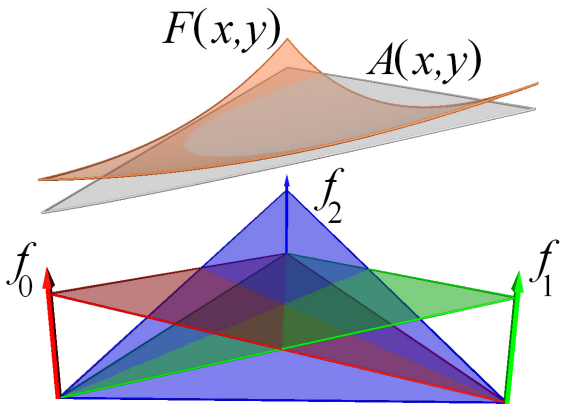

Figure 2.4: Best-approximation example. Orange surface shows the original function $F(x, y)$. Gray surface shows the linear spline approximation $A(x, y) = c_0 f_0 + c_1 f_1 + c_2 f_2$. Basis functions f_0, f_1, and f_2 are shown for knot locations $\mathbf{k}_0$, $\mathbf{k}_1$, and $\mathbf{k}_2$, respectively.

Change-of-variables theorem

Let $\mathbb{R}$ and $\mathbb{U}$ be regions in the plane and let $M : \mathbb{U} \longrightarrow \mathbb{R}$ be a C^1-continuous one-to-one mapping such that $M(\mathbb{U}) = \mathbb{R}$. Then, for any bivariate integrable function $f(x, y)$, the equation

$$\int_{\mathbb{R}} f(x, y)dxdy = \int_{\mathbb{U}} f(x(u, v), y(u, v))Jdudv \tag{2.12}$$

holds, where J is the Jacobian of M,

$$J = \det \begin{bmatrix} \frac{\partial}{\partial u}x(u, v) & \frac{\partial}{\partial v}x(u, v) \\ \frac{\partial}{\partial u}y(u, v) & \frac{\partial}{\partial v}y(u, v) \end{bmatrix}. \tag{2.13}$$

Thus, it is possible to effectively compute an integral over a triangle T having vertices $\mathbf{k}_0 = (x_0, y_0)^{\mathrm{T}}$, $\mathbf{k}_1 = (x_1, y_1)^{\mathrm{T}}$, and $\mathbf{k}_2 = (x_2, y_2)^{\mathrm{T}}$, by mapping the standard triangle $T^{\mathbb{U}}$ to T using the linear transformation

$$\begin{bmatrix} x(u, v) \\ y(u, v) \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} x_0 \\ y_1 \end{bmatrix}. \tag{2.14}$$

This transformation maps the standard triangle $T^{\mathbb{U}}$ with vertices $\mathbf{u}_0 = (0, 0)^{\mathrm{T}}$, $\mathbf{u}_1 = (1, 0)^{\mathrm{T}}$, and $\mathbf{u}_2 = (0, 1)^{\mathrm{T}}$ in the uv-plane to the arbitrary triangle T in the xy-plane. For this linear mapping, the change-of-variables theorem yields

$$\int_T f(x, y)dxdy = J \int_{v=0}^{1} \int_{u=0}^{1-v} f\left(x(u, v), y(u, v)\right) dudv, \tag{2.15}$$

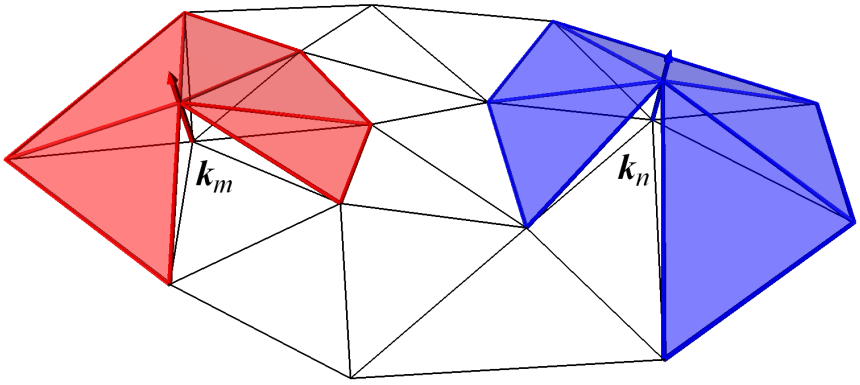

Figure 2.5: Platelets of $\mathbf{k}_m$ and $\mathbf{k}_n$ and associated basis functions. (Front platelet triangles have been removed for clarity.)

and the Jacobian is given by

$$J = \det \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}. \tag{2.16}$$

As in the univariate case, the basis functions $f_i(x, y)$ are hat functions. Thus, $f_i(x, y)$ has value one at knot $\mathbf{k}_i$, varies linearly to zero as one approaches its neighbors, and is zero everywhere else. The only scalar products one must consider are $\langle N_0, N_0 \rangle$ and $\langle N_0, N_1 \rangle$, where $N_i = \mathbf{u}_i = (u_i, v_i)^{\mathrm{T}}$ is a linear spline basis function defined over the standard triangle $T^{\mathbb{U}}$. The values of these two scalar products are

$$\langle N_0, N_0 \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} (1 - u - v)^2 \, dudv = \frac{1}{12} \tag{2.17}$$

and

$$\langle N_0, N_1 \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} (1 - u - v)u \, dudv = \frac{1}{24}. \tag{2.18}$$

The area of influence for the basis function f_i is the platelet of triangles connected to knot $\mathbf{k}_i$, see Figure 2.5. The function f_i varies linearly from one to zero over all triangles defining $\mathbf{k}_i$'s platelet. The scalar product $\langle f_m, f_m \rangle$ is given by

$$\langle f_m, f_m \rangle = \sum_{j=0}^{n_m - 1} \int_{T_j} f_m(x, y) f_m(x, y) dxdy = \frac{1}{12} \sum_{j=0}^{n_m - 1} J_j, \tag{2.19}$$

where n_m is the number of triangles in the platelet of knot $\mathbf{k}_m$ and J_j is the Jacobian associated with the j^{th} platelet triangle T_j. (The platelet of knot $\mathbf{k}_m$ is the set of triangles

$\mathcal{T}_m = \{T_j\},\ 0 \le j \le n_m - 1.$) The scalar product $\langle f_m, f_n \rangle$ whose associated knots $\mathbf{k}_m$ and $\mathbf{k}_n$ are connected by and edge is given by

$$\langle f_m, f_n \rangle = \sum_{j=0}^{n_{m,n}-1} \int_{T_j} f_m(x,y) f_n(x,y) dxdy = \frac{1}{24} \sum_{j=0}^{n_{m,n}-1} J_j, \tag{2.20}$$

where $n_{m,n}$ is the number of triangles in the set $\mathcal{T}_{m,n}$, which is the intersection of sets $\mathcal{T}_m$ and $\mathcal{T}_n$. Mapping the knots $\mathbf{k}_m$ and $\mathbf{k}_n$ to $\mathbf{u}_0$ and $\mathbf{u}_1$, respectively, always maps the standard triangle $T^{\mathbb{U}}$ to triangle $T_j \in \mathcal{T}_{m,n},\ 0 \le j \le n_{m,n} - 1$, thus, the reason why one only needs to consider the scalar product $\langle N_0, N_1 \rangle$.

The global error E for the linear spline approximation $A(x,y)$, defined by triangulation $\mathcal{T}$ containing N_T triangles, is computed using the L_2 norm of $D(x,y) = F(x,y) - A(x,y)$, the difference between $A(x,y)$ and $F(x,y)$, given as

$$E = \langle D, D \rangle = \sqrt{\int_{\mathcal{T}} \Big(F(x,y) - A(x,y) \Big)^2 dxdy}\ . \tag{2.21}$$

The local error e_i for triangle $T_i \in \mathcal{T},\ i = 0,\ \ldots,\ N_T - 1$, is defined as

$$e_i = \sqrt{\int_{T_i} \left(F(x,y) - T_i \right)^2 dx}. \tag{2.22}$$

Integrals in the 2D case are computed using Romberg integration, which is discussed in the Appendix, see Section A.2. An approximation $A(x,y)$ of a height field of Crater Lake data set is shown in Figure 2.6. The original data consists of 158346 data sites. The approximation $A(x,y)$ has 93 knots, 159 triangles, and an error of 8.62 based on the sum of the local errors $\sum_{i=0}^{N_T-1} e_i$.

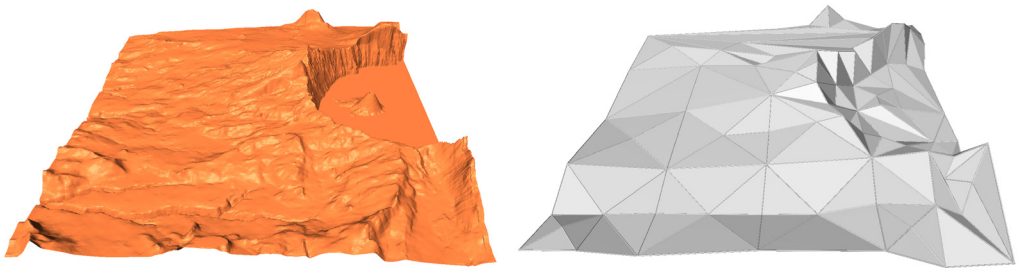

Figure 2.6: Bivariate approximation of Crater Lake data set. Left image shows the original data set $F(x,y)$. Right image shows the approximation $A(x,y)$ having 93 knots and 159 triangles. The approximation has an error of 8.62.

2.3 The 3D Case

The 3D case is a straightforward generalization of the 2D case. This is done by replacing the notion of a triangle with that of a tetrahedron. Hat functions are again used as the basis functions and the best-approximation method is applied to find the coefficients c_i at the knots $\mathbf{k}_i$.

A linear spline is defined by a set of N_K knots $\mathbf{k}_i = (x_i, y_i, z_i)^{\mathrm{T}} \in \mathbb{R}^3,\ 0 \leq i \leq N_K - 1$, coefficients c_i positioned at these knot locations, and a set $\mathcal{T}$ of N_T knot quadruples defining the tetrahedralization of the knots. Spline points $\mathbf{p}_i = (x_i, y_i, z_i, c_i)^{\mathrm{T}}$ are formed from the knot locations and coefficients.

Only the implications of the change-of-variables theorem are discussed, which is important in the context of computing the required scalar products and error estimates defined over tetrahedral domains. The mapping of the standard tetrahedron $T^{\mathbb{U}}$—with vertices $\mathbf{u}_0 = (0,0,0)^{\mathrm{T}}$, $\mathbf{u}_1 = (1,0,0)^{\mathrm{T}}$, $\mathbf{u}_2 = (0,1,0)^{\mathrm{T}}$, and $\mathbf{u}_3 = (0,0,1)^{\mathrm{T}}$—in uvw-parameter space to an arbitrary tetrahedron T in xyz-physical space—having vertices $\mathbf{k}_0 = (x_0, y_0, z_0)^{\mathrm{T}}$, $\mathbf{k}_1 = (x_1, y_1, z_1)^{\mathrm{T}}$, $\mathbf{k}_2 = (x_2, y_2, z_2)^{\mathrm{T}}$, and $\mathbf{k}_3 = (x_3, y_3, z_3)^{\mathrm{T}}$—is given by the linear transformation

$$\begin{bmatrix} x(u,v,w) \\ y(u,v,w) \\ z(u,v,w) \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}. \tag{2.23}$$

In this case, the change-of-variables theorem implies that

$$\int_T f(x,y,z)\ dxdydz = J \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} f\Big(x(u,v,w), y(u,v,w), z(u,v,w)\Big)\ dudvdw, \tag{2.24}$$

where the Jacobian is given as

$$J = \det \begin{bmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{bmatrix}. \tag{2.25}$$

Using the same argument as in the 2D case, the only scalar products one needs to consider are $\langle N_0, N_0 \rangle$ and $\langle N_0, N_1 \rangle$, where $N_i = \mathbf{u}_i = (u_i, v_i, w_i)^{\mathrm{T}}$ is a linear spline basis

function defined over the standard tetrahedron $T^{\mathbb{U}}$. The values of these two scalar products are

$$\langle N_0, N_0 \rangle = \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} (1-u-v-w)^2 \; dudvdw = \frac{1}{60} \tag{2.26}$$

and

$$\langle N_0, N_1 \rangle = \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} (1-u-v-w)u \; dudvdw = \frac{1}{120}. \tag{2.27}$$

Integrals in the 3D case are computed using Romberg integration, which is discussed in the Appendix, see Section A.3.

2.4 Constructing Hierarchical Approximations

For practical visualization of large data sets one must compute a hierarchy of approximations to represent the data. In general, this hierarchy should consists of a low-resolution approximation and several increasingly better approximations. For spline approximations, such as the ones discussed here, a low-resolution approximation contains a relatively small number of approximation elements (i.e., intervals, triangles, or tetrahedra) and higher-resolution approximations contain progressively more elements. For visualization purposes, one can extract a *level* from the hierarchy that meets desired specifications based on network traffic, available memory, machine speed, etc.

The hierarchy of approximations created here are constructed using a refinement method. One begins with a low-resolution level and refines this mesh—by subdividing the element having the highest error—to construct the next level in the hierarchy. The basic steps of the method are

1. **Initial approximation.** Define an initial, coarse decomposition of the function's domain and, for all knots, compute the coefficients defining the best linear spline approximation.

2. **Error estimation.** Analyze the error of this approximation by computing appropriate global and local error estimates.

3. **Refinement.** Identify the element with maximal local error estimate and subdivide it.

4. **Recompute approximation.** Recompute the best linear spline approximation based on the updated knot set and decomposition.

5. **Iteration.** Repeat steps 2, 3, and 4 until a user-specified error tolerance is satisfied.

A refinement method constructs a hierarchy "bottom up"—from lowest resolution to highest resolution. A method that constructs a hierarchy "top down" is called a simplification method. Thus, it begins with a high-resolution representation of the data and removes elements (or data) to construct various lower-resolution levels in a hierarchy. A few examples of simplification methods are described in [7, 8, 16, 17, 20, 27, 28].

There are several refinement strategies and methods, some are described in [19, 39, 45]. Longest-edge bisection is used here and the details for the 1D, 2D, and 3D cases are described in the following sections. In all cases, the element to be subdivided is bisected by inserting a new knot at the midpoint of its longest edge. Methods that consider variable knot locations are described in [6, 41, 42].

Alternatively, one may want to use a refinement scheme other than longest-edge bisection. This is not an issue in the 1D case, since there aren't very many possibilities when subdividing, however, there may be advantages in the 2D and 3D cases. A scheme such as red-green splitting, see [19], ensures well-shaped elements that are all similar to the elements that define the coarsest level of the hierarchy. Thus, no chance of long and skinny elements. Another method that does not produce skinny elements is diamond subdivision, see [18].

2.4.1 The 1D Case

Longest-edge bisection, in this case, reduces to bisecting an interval by inserting a new knot at the midpoint of the interval. Successive levels in a hierarchy are created by inserting one knot at the midpoint of the segment having the largest local error at each iteration. Five levels from a hierarchy of approximations of $F(x) = 10x\left(x - \frac{1}{2}\right)\left(x - \frac{3}{4}\right),\ 0 \leq x \leq 1$, are shown in Figure 2.7. The initial decomposition is a single line segment, having two knots, that covers the interval $[0, 1]$. One can easily see how the addition of knots to the spline improves the overall approximation.

2.4.2 The 2D Case

In this case, one must subdivide triangles in order to create a hierarchy of approximations. To subdivide a triangle, one must determine the longest edge and insert a knot at the midpoint of this edge. Incidently, one must also induce splits in neighboring elements that share the split edge to guarantee a mesh that does not have "hanging nodes." (In the 2D case, either no or one edge neighbor exists.) Figure 2.8 shows a sample hierarchy that approximates $F(x, y) = 10x\left(x - \frac{1}{4}\right)\left(x - \frac{3}{4}\right)y^2,\ 0 \leq x, y \leq 1$. The subdivision tends to focus on high-gradient areas, since the local error estimates in these regions are relatively high. In general, the concept of concentrating elements in high-gradient regions is intuitive because more elements are needed to better represent these regions, since the function being approximated changes dramatically in these areas.

When computing the hierarchy, one can at each iteration choose to refine multiple elements simultaneously. This more quickly refines the approximation and more granular levels are produced in the hierarchy. All of the 2D and 3D examples shown refine the top ten percent of highest-error elements at each iteration to improve performance. Additionally, since many of elements of the best-approximation normal equations do not change, one can reuse matrix elements to further improve performance.

2.4.3 The 3D Case

To subdivide a tetrahedron, one must determine the longest edge and insert a knot at the midpoint of this edge. Incidently, as in the 2D case, one must also induce splits in neighboring elements that share the split edge to guarantee a mesh that does not have hanging nodes. However, unlike the 2D case, there can be any number of edge neighbors. Figure 2.9 shows a sample hierarchy approximating skull data. Two cutting planes are used to "slice" through the 3D data domain. Light areas show high-density regions and dark areas show low-density regions.

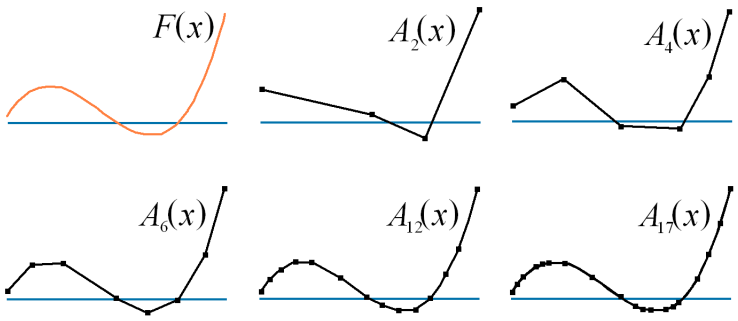

Figure 2.7: Hierarchical approximation example. Top-left image shows the original function $F(x) = 10x\left(x - \frac{1}{2}\right)\left(x - \frac{3}{4}\right),\ 0 \le x \le 1$. Top-middle to bottom-right images show increasingly better approximations $A_l(x)$, where l is the level in the hierarchy. Approximations contain 4, 6, 8, 14, and 19 knots, respectively.

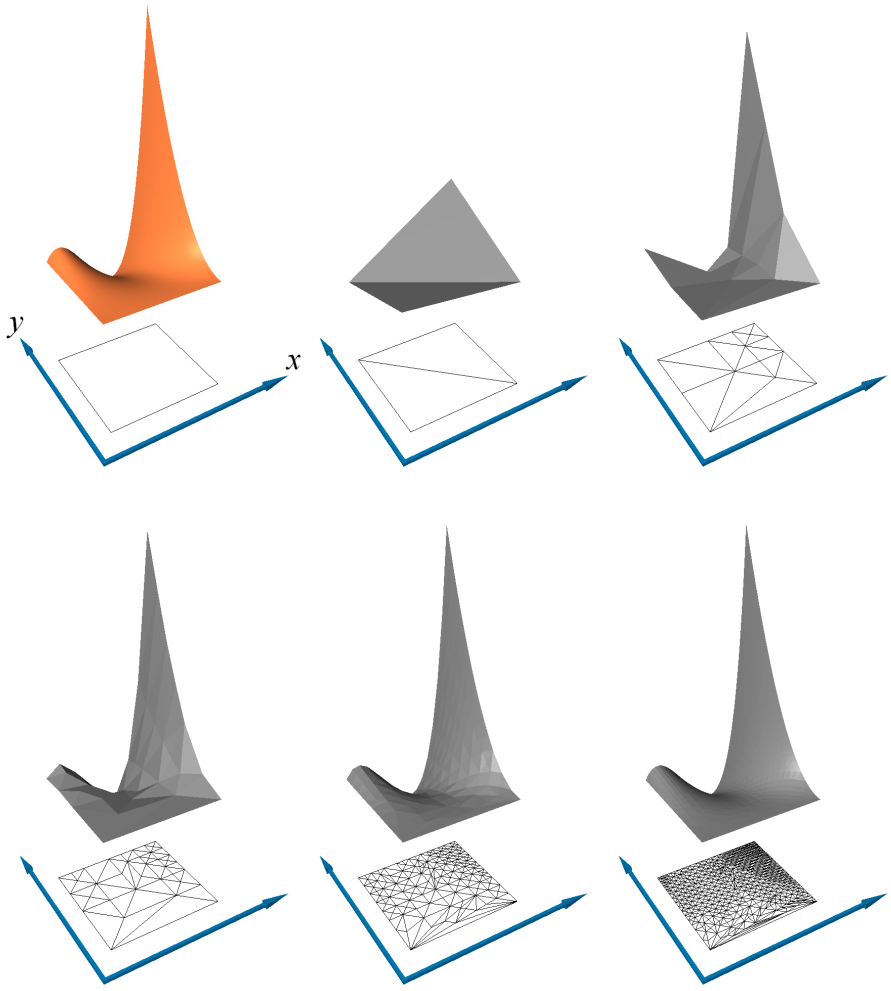

Figure 2.8: Top-left image shows the original function $F(x, y) = 10x\left(x - \frac{1}{4}\right)\left(x - \frac{3}{4}\right)y^2,\ 0 \le x, y \le 1$. Top-middle to bottom-right images show increasingly better approximations. Triangulation for each approximation is shown in the xy-plane. Approximations contain 4, 14, 38, 162, and 859 knots and have error estimates of 3.2×10^{-2}, 9.3×10^{-4}, 8.3×10^{-5}, 3.7×10^{-6}, and 1.1×10^{-7}, respectively

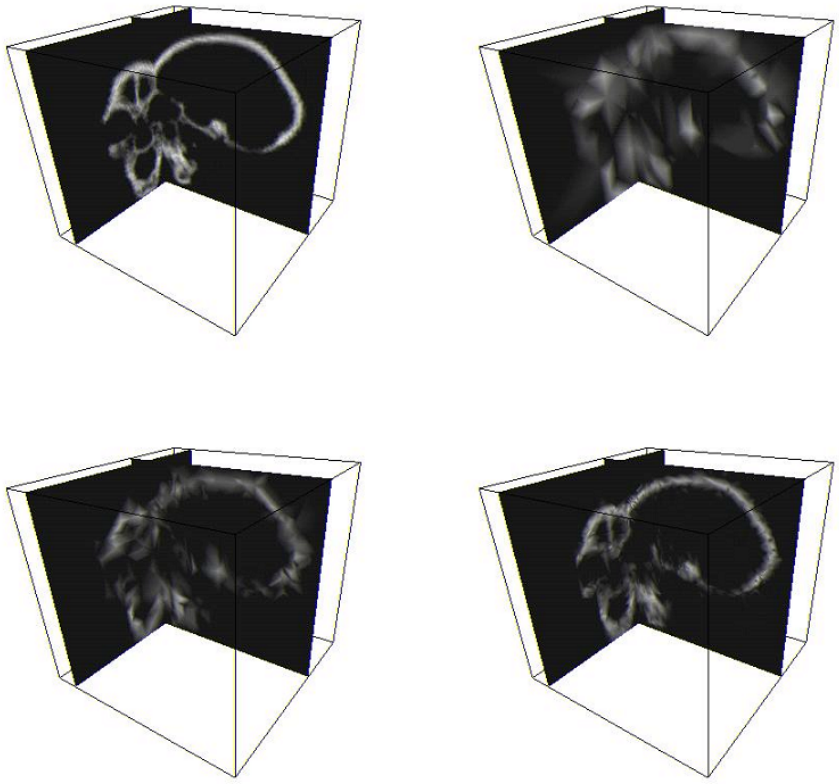

Figure 2.9: Three approximations of skull data (upper-right to lower-right images). Upper-left image shows the original data, containing 278528 data sites. The approximations contain 326, 1775, and 9776 knots and have an error of 0.1087, 0.0794, and 0.0399, respectively.

2.5 Finite Element Approach

Under certain conditions, one may be able to conserve storage space by only inserting knots at discrete locations. Since many data sets have implied data site locations, it may be possible to reduce the required storage space by "smart" or implied indexing of the knots (based on the original data sites) in an approximation. Consider the case of approximating a rectilinear grid where all the data site locations are implied. With clever knot indexing, the required storage space for an approximation can be significantly reduced. This improvement is easily incorporated into the refinement process that generates hierarchies. When inserting a new knot, one simply "snaps" its location to the nearest originally provided discrete location. Figure 2.10 shows a 1D example of the knot insertion process.

When "snapping," one must take care that the found knot location is inside the domain of the element being refined. In all cases (1D, 2D, and 3D), malformed elements may

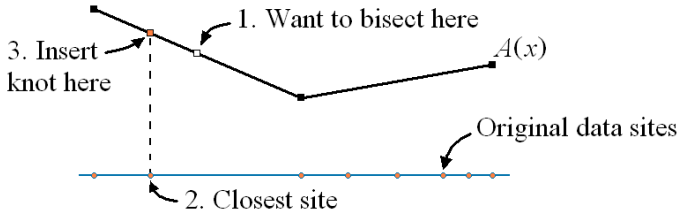

Figure 2.10: Insert new knots at discrete locations only. 1) When subdividing an element, one wants to bisect at the midpoint of the chosen element. 2) The nearest discrete location is found. 3) The inserted knot is "snapped" to the found discrete location.

be constructed by snapping to a knot location that is outside of the domain of the element. This causes serious problems in the approximation process and must be avoided. Additionally, snapping to one of the element boundaries—other than the edge being bisected—should also be avoided. Figure 2.11 shows a malformed triangle formation caused when the midpoint of the longest edge is snapped to a knot location outside of the domain of the triangle being refined.

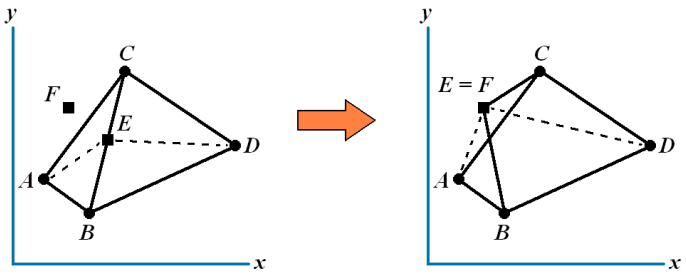

Figure 2.11: Example of malformed triangles when the midpoint E of the longest edge (formed by B and C) is snapped to a knot location F outside of the domain of the triangle(s) being refined.

2.6 Using First-derivative Information

In some applications, it may be appropriate to require an approximation to respect the extrema of the function being approximated. One example is when approximating a sound wave. Distortion may occur if *over-* and *under-shoots* are present, since the minimum and maximum amplitude values may be to low (or high) to be reproduced faithfully. Examples of over- and under-shoots are shown in Figure 2.12. Over- and under-shoots can be reduced by including first-derivative information in the computation of the spline coefficients. The

method described here is discussed in [44].

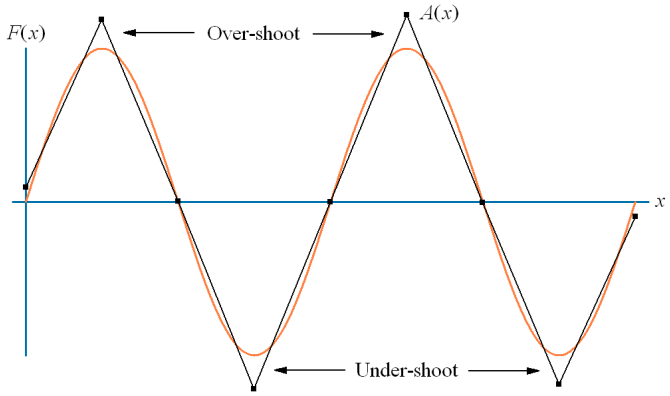

Figure 2.12: Demonstration of over- and under-shoots in approximation. The orange curve is the original function $F(x) = \sin(4\pi x)$, $0 \leq x \leq 1$. Dark spline is the approximation $A(x)$, having 9 knots.

2.6.1 The 1D Case

First-derivative information is easily integrated into the best-approximation method. One need only modify the definition of the scalar product $\langle g, h \rangle$ of two functions $g(x)$ and $h(x)$, defined over the interval $[a, b]$ as

$$\langle g, h \rangle = \int_a^b w_0 g(x)h(x) + w_1 g'(x)h'(x)dx, \tag{2.28}$$

where the "weights" w_0 and w_1 are user-specified, such that $w_0 > 0$, $w_1 \geq 0$, and $w_0 + w_1 = 1$. A *Sobolev*-like L_2 norm [29] for function $D(x) = F(x) - A(x)$ is used to measure the error of an approximation and is defined as

$$\sqrt{\langle D, D \rangle} = \sqrt{\int_a^b w_0 D(x)^2 + w_1 D'(x)^2 dx}. \tag{2.29}$$

Revising the best-approximation normal equations $M\mathbf{c} = \mathbf{F}$, see Equation (2.1), to use the more general scalar product, which contains first-derivative information, one obtains the elements $a_{i,j}$ of M from

$$a_{i,j} = w_0 \int_a^b f_i(x)f_j(x)dx + w_1 \int_a^b f_i'(x)f_j'(x)dx, \ 0 \leq i, j \leq N-1, \tag{2.30}$$

and the elements l_i of $\mathbf{F}$ are given by

$$l_i = w_0 \int_a^b F(x)f_i(x)dx + w_1 \int_a^b F'(x)f_i'(x)dx, \ 0 \leq i \leq N-1. \tag{2.31}$$

Integral values required to compute the matrix elements $a_{i,j}$ are

$$\int_{k_0}^{k_1}(f_0(x))^2dx = \frac{1}{3}\Delta_0, \tag{2.32}$$

$$\int_{k_{N-2}}^{k_{N-1}}(f_{N-2}(x))^2dx = \frac{1}{3}\Delta_{N-2}, \tag{2.33}$$

$$\int_{k_{i-1}}^{k_{i+1}}(f_i(x))^2dx = \frac{1}{3}(\Delta_{i-1}+\Delta_i),\ 1\le i\le N-2,\ \text{and} \tag{2.34}$$

$$\int_{k_i}^{k_{i+1}}f_i(x)f_{i+1}(x)dx = \frac{1}{6}\Delta_i,\ 0\le i\le N-2. \tag{2.35}$$

Terms involving the first derivative are

$$\int_{k_0}^{k_1}(f_0'(x))^2dx = \frac{1}{\Delta_0}, \tag{2.36}$$

$$\int_{k_{N-2}}^{k_{N-1}}(f_{N-2}'(x))^2dx = \frac{1}{\Delta_{N-2}}, \tag{2.37}$$

$$\int_{k_{i-1}}^{k_{i+1}}(f_i'(x))^2dx = \frac{1}{\Delta_{N-1}}+\frac{1}{\Delta_i},\ 1\le i\le N-2,\ \text{and} \tag{2.38}$$

$$\int_{k_i}^{k_{i+1}}f_i'(x)f_{i+1}'(x)dx = \frac{-1}{\Delta_i},\ 0\le i\le N-2. \tag{2.39}$$

Thus, the tridiagonal matrix M can be described as the "weighted sum" of two tridiagonal matrices M_0 and M_1, given by

$$M=\frac{1}{6}(w_0M_0+w_1M_1), \tag{2.40}$$

where the matrices M_0 and M_1 are given by

$$M_0=\begin{bmatrix} 2\Delta_0 & \Delta_0 & & & & & \\ \Delta_0 & 2(\Delta_0+\Delta_1) & \Delta_1 & & & & \\ & \Delta_1 & 2(\Delta_1+\Delta_2) & \Delta_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & \Delta_{N-2} & 2\Delta_{N-2} \end{bmatrix}, \tag{2.41}$$

and

$$M_1=6\begin{bmatrix} \frac{1}{\Delta_0} & \frac{1}{\Delta_0} & & & & & \\ \frac{-1}{\Delta_0} & \frac{\Delta_0+\Delta_1}{\Delta_0\Delta_1} & \frac{-1}{\Delta_1} & & & & \\ & \frac{-1}{\Delta_1} & \frac{\Delta_1+\Delta_2}{\Delta_1\Delta_2} & \Delta_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & \frac{-1}{\Delta_{N-2}} & \frac{1}{\Delta_{N-2}} \end{bmatrix}. \tag{2.42}$$

A 1D approximation using first-derivative information is shown in Figure 2.13. In this example, the function $F(x) = \sin(4\pi x)$, $0 \leq x \leq 1$, is approximated by a spline having 9 knots uniformly spaced across the interval $[0, 1]$. A few different combinations of weights are used to illustrate the effect of first-derivative information on the approximation. Even the slightest addition of first-derivative information effects the resulting approximation dramatically.

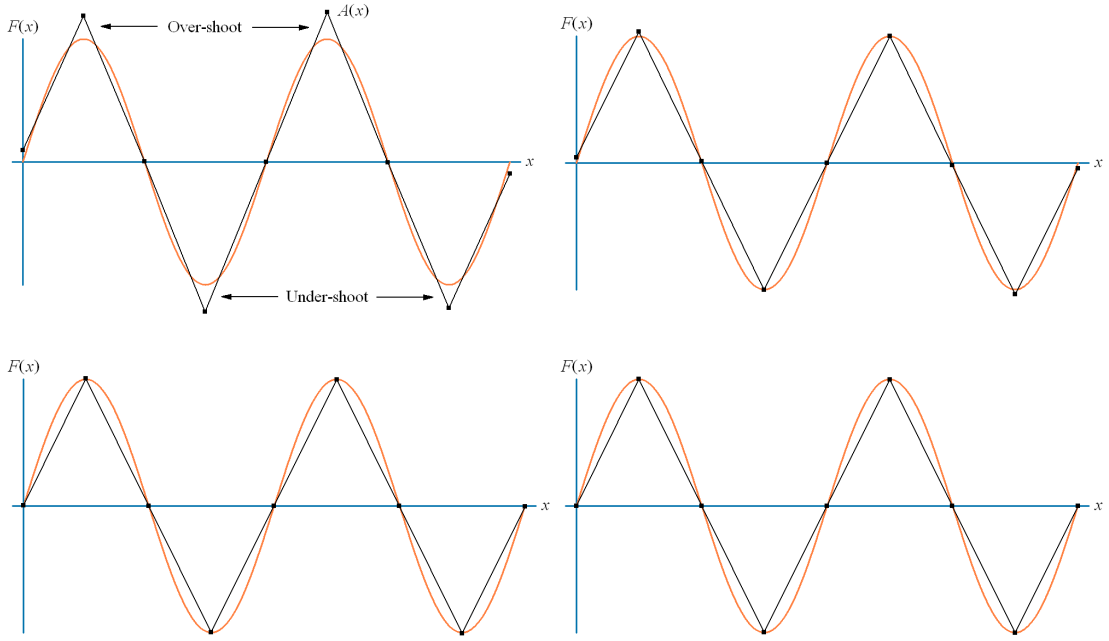

Figure 2.13: Top-left image shows the original approximation $A(x)$ of function $F(x) = \sin(4\pi x)$, $0 \leq x \leq 1$. In each image, the black spline shows the approximation $A(x)$ and the orange curve shows the original function $F(x)$. Top-left image uses weights $(1.0, 0.0)$, where the weights are given as (w_0, w_1). Top-right image uses weights $(0.9, 0.1)$. Bottom-left image uses weights $(0.5, 0.5)$. Bottom-right image uses weights $(0.1, 0.9)$.

2.6.2 The 2D Case

Using the generalization of Equation (2.28), the scalar product $\langle g, h \rangle$ of two functions $g(x, y)$ and $h(x, y)$, defined over triangle T, is given as

$$\langle g, h \rangle = \int_T w_{0,0} g(x,y) h(x,y) + w_{1,0} g^x(x,y) h^x(x,y) + w_{0,1} g^y(x,y) h^y(x,y) dxdy, \qquad (2.43)$$

where, for an arbitrary function $s(x, y)$, $s^i(x, y)$ corresponds to the partial derivative of function $s(x, y)$ with respect to $i \in \{x, y\}$. The weights in this case are $w_{0,0} > 0$, $w_{1,0}, w_{0,1} \geq$

0, and $w_{0,0} + w_{1,0} + w_{0,1} = 1$, however, in general, $w_{1,0} = w_{0,1}$ so that equal weights are applied to each direction of the derivative.

Revising the best-approximation normal equations $M\mathbf{c} = \mathbf{F}$, see Equation (2.1), to use the more general scalar product, which contains first-derivative information, one obtains the elements $a_{i,j}$ of M from

$$\begin{aligned} a_{i,j} &= w_{0,0} \int_T f_i(x,y) f_j(x,y) dxdy + \\ & \quad w_{1,0} \int_T f_i^x(x,y) f_j^x(x,y) dxdy + \\ & \quad w_{0,1} \int_T f_i^y(x,y) f_j^y(x,y) dxdy, \ 0 \le i,j \le N-1, \end{aligned} \tag{2.44}$$

and the elements l_i of $\mathbf{F}$ are given by

$$\begin{aligned} l_i &= w_{0,0} \int_T F(x,y) f_i(x,y) dxdy + \\ & \quad w_{1,0} \int_T F^x(x,y) f_i^x(x,y) dxdy + \\ & \quad w_{0,1} \int_T F^y(x,y) f_i^y(x,y) dxdy, \ 0 \le i \le N-1. \end{aligned} \tag{2.45}$$

Integral values to compute the matrix elements $a_{i,j}$ are

$$\sum_{j=0}^{n_m-1} \int_{T_j} (f_m(x,y))^2 dxdy = \frac{1}{12} \sum_{j=0}^{n_m-1} J_j, \tag{2.46}$$

where n_m is the number of triangles in the platelet of knot $\mathbf{k}_m$ and J_j is the Jacobian associated with the j^{th} platelet triangle. (The platelet of knot $\mathbf{k}_m$ is the set of triangles $\mathcal{T}_m = \{T_j\}, \ 0 \le j \le n_m - 1$.) An integral value required to compute the scalar product $\langle f_m, f_n \rangle$ whose associated knots $\mathbf{k}_m$ and $\mathbf{k}_n$ are connected by and edge is given by

$$\sum_{j=0}^{n_{m,n}-1} \int_{T_j} f_m(x,y) f_n(x,y) dxdy = \frac{1}{24} \sum_{j=0}^{n_{m,n}-1} J_j, \tag{2.47}$$

where $n_{m,n}$ is the number of triangles in the set $\mathcal{T}_{m,n}$, which is the intersection of sets $\mathcal{T}_m$ and $\mathcal{T}_n$. The linear polynomial $f(x,y)$ interpolating the values one, zero, and zero at the vertices $(x_0, y_0)^{\mathrm{T}}$, $(x_1, y_1)^{\mathrm{T}}$, and $(x_2, y_2)^{\mathrm{T}}$, respectively, has the partial derivatives

$$f^x(x,y) = -\frac{1}{J} \det \begin{bmatrix} 1 & y_1 \\ 1 & y_2 \end{bmatrix} = \frac{y_1 - y_2}{J} \tag{2.48}$$

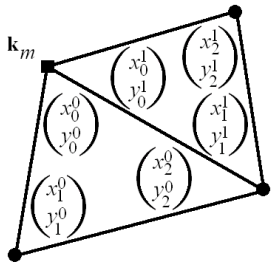

Figure 2.14: Indexing scheme for platelet knots relative to $\mathbf{k}_m$ in 2D case (neighboring triangles oriented counterclockwise).

and

$$f^y(x,y) = -\frac{1}{J}\det\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} = \frac{x_2 - x_1}{J}. \tag{2.49}$$

Integrals involving these partial derivatives are

$$\sum_{j=0}^{n_m-1}\int_{T_j}\left(f_m^x(x,y)\right)^2 dxdy = \frac{1}{2}\sum_{j=0}^{n_m-1}\frac{1}{J_j}\left(\det\begin{bmatrix} 1 & y_1^j \\ 1 & y_2^j \end{bmatrix}\right)^2, \tag{2.50}$$

and

$$\sum_{j=0}^{n_m-1}\int_{T_j}\left(f_m^y(x,y)\right)^2 dxdy = \frac{1}{2}\sum_{j=0}^{n_m-1}\frac{1}{J_j}\left(\det\begin{bmatrix} x_1^j & 1 \\ x_2^j & 1 \end{bmatrix}\right)^2, \tag{2.51}$$

where $(x_0^j, y_0^j)^{\mathrm{T}}$, $(x_1^j, y_1^j)^{\mathrm{T}}$, and $(x_2^j, y_2^j)^{\mathrm{T}}$ are the counterclockwise-ordered vertices of the j^{th} platelet triangle associated with knot $\mathbf{k}_m$, see Figure 2.14. Other required values are

$$\sum_{j=0}^{n_{m,n}-1}\int_{T_j} f_m^x(x,y) f_n^x(x,y) dxdy = \frac{1}{2}\sum_{j=0}^{n_{m,n}-1}\frac{1}{J_j}\det\begin{bmatrix} 1 & y_1^j \\ 1 & y_2^j \end{bmatrix}\det\begin{bmatrix} 1 & y_0^j \\ 1 & y_1^j \end{bmatrix} \tag{2.52}$$

and

$$\sum_{j=0}^{n_{m,n}-1}\int_{T_j} f_m^y(x,y) f_n^y(x,y) dxdy = \frac{1}{2}\sum_{j=0}^{n_{m,n}-1}\frac{1}{J_j}\det\begin{bmatrix} x_1^j & 1 \\ x_2^j & 1 \end{bmatrix}\det\begin{bmatrix} x_0^j & 1 \\ x_1^j & 1 \end{bmatrix}, \tag{2.53}$$

where $n_{m,n}$ is the number of platelet triangles in the set $\mathcal{T}_{m,n}$—the common platelet triangles between knots $\mathbf{k}_m$ and $\mathbf{k}_n$—and $(x_0^j, y_0^j)^{\mathrm{T}}$, $(x_1^j, y_1^j)^{\mathrm{T}}$, and $(x_2^j, y_2^j)^{\mathrm{T}}$ are vertices of a triangle $T_j \in \mathcal{T}_{m,n}$, see Figure 2.15.

A 2D approximation using first-derivative information is shown in Figure 2.16. A "checkerboard function" was digitized to a 100×100 grid to which a linear spline was fit.

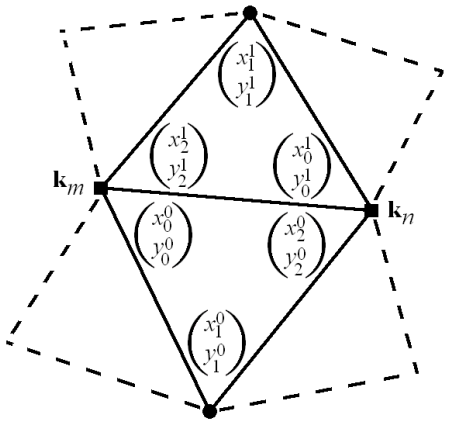

Figure 2.15: Indexing scheme for platelet knots relative to $\mathbf{k}_m$ and $\mathbf{k}_n$ in 2D case (neighboring triangles oriented counterclockwise).

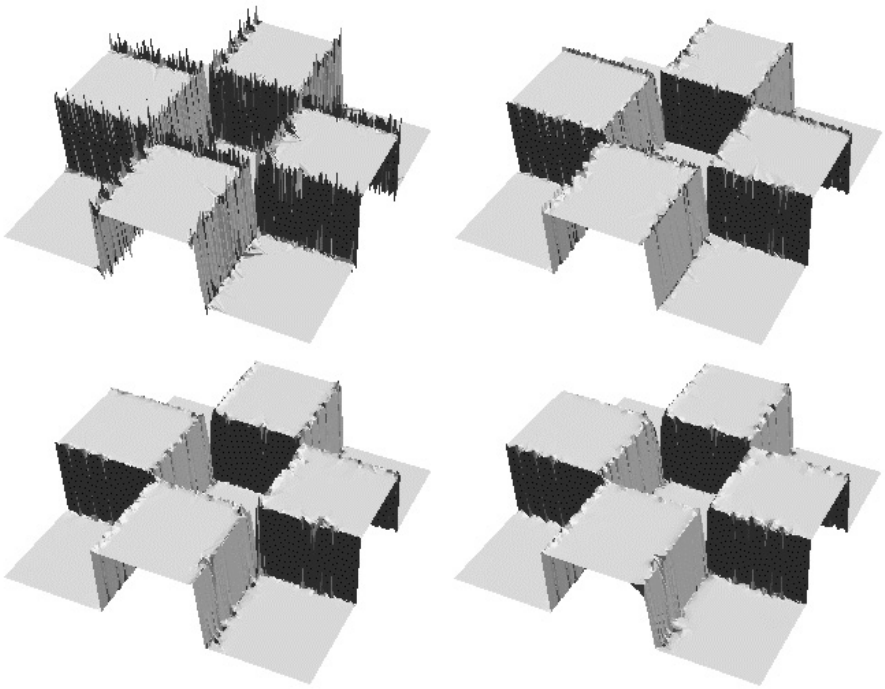

Figure 2.16: Four approximations of 2D checkerboard function with varying weights $(w_{0,0}, w_{1,0}, w_{0,1})$: $(1,0,0)$, $(\frac{3}{4},\frac{1}{8},\frac{1}{8})$, $(\frac{1}{2},\frac{1}{4},\frac{1}{4})$, and $(\frac{1}{4},\frac{3}{8},\frac{3}{8})$, from upper-left to lower-right corner, number of knots varying between 5000 and 6000.

The approximations were computed for this spline using first-derivative information and the finite-element approach described in Section 2.5. It is obvious in this example that the first-derivative affects the over- and under-shoots significantly. As more weight is added to the first-derivative information, the better the approximation becomes.

2.6.3 The 3D Case

Generalizing to the 3D case, the scalar product $\langle g, h \rangle$ of two functions $g(x, y, z)$ and $h(x, y, z)$, defined over tetrahedron T, is defined as

$$\begin{aligned} \langle g, h \rangle &= \int_T w_{0,0,0} g(x,y,x) h(x,y,x) + \\ & \quad w_{1,0,0} g^x(x,y,z) h^x(x,y,z) + \\ & \quad w_{0,1,0} g^y(x,y,z) h^y(x,y,z) + \\ & \quad w_{0,0,1} g^z(x,y,z) h^z(x,y,z) dxdydz, \end{aligned} \tag{2.54}$$

where, for an arbitrary function $s(x, y, z)$, $s^i(x, y, z)$ corresponds to the partial derivative of function $s(x, y, z)$ with respect to $i \in \{x, y, z\}$. The weights in this case are $w_{0,0,0} > 0$, $w_{1,0,0}, w_{0,1,0}, w_{0,0,1} \geq 0$, and $w_{0,0,0} + w_{1,0,0} + w_{0,1,0} + w_{0,0,1} = 1$, however, in general, $w_{1,0,0} = w_{0,1,0} = w_{0,0,1}$ so that equal weights are applied to each direction of the derivative.

Revising the best-approximation normal equations $M\mathbf{c} = \mathbf{F}$, see Equation (2.1), to use the more general scalar product, which contains first-derivative information, one obtains the elements $a_{i,j}$ of M from

$$\begin{aligned} a_{i,j} &= w_{0,0,0} \int_T f_i(x,y,z) f_j(x,y,z) dxdydz + \\ & \quad w_{1,0,0} \int_T f_i^x(x,y,z) f_j^x(x,y,z) dxdydz + \\ & \quad w_{0,1,0} \int_T f_i^y(x,y,z) f_j^y(x,y,z) dxdydz + \\ & \quad w_{0,0,1} \int_T f_i^z(x,y,z) f_j^z(x,y,z) dxdydz, \ 0 \leq i,j \leq N-1, \end{aligned} \tag{2.55}$$

and the elements l_i of $\mathbf{F}$ are given by

$$\begin{aligned} l_i &= w_{0,0,0} \int_T F(x,y,z) f_i(x,y,z) dxdydz + \\ & \quad w_{1,0,0} \int_T F^x(x,y,z) f_i^x(x,y,z) dxdydz + \end{aligned}$$

$$w_{0,1,0}\int_T F^y(x,y,z)f_i^y(x,y,z)dxdydz +$$
$$w_{0,0,1}\int_T F^z(x,y,z)f_i^z(x,y,z)dxdydz,\ 0\le i\le N-1. \tag{2.56}$$

Integral values to compute the matrix elements $a_{i,j}$ are

$$\sum_{j=0}^{n_m-1}\int_{T_j}\Big(f_m(x,y,z)\Big)^2 dxdydz = \frac{1}{60}\sum_{j=0}^{n_m-1} J_j, \tag{2.57}$$

where n_m is the number of tetrahedra in the platelet of knot $\mathbf{k}_m$ and J_j is the Jacobian associated with the j^{th} platelet tetrahedron. (The platelet of knot $\mathbf{k}_m$ is the set of tetrahedra $\mathcal{T}_m = \{T_j\},\ 0\le j\le n_m-1$.) An integral value required to compute the scalar product $\langle f_m, f_n\rangle$ whose associated knots $\mathbf{k}_m$ and $\mathbf{k}_n$ are connected by and edge is given by

$$\sum_{j=0}^{n_{m,n}-1}\int_{T_j} f_m(x,y,z)f_n(x,y)dxdydz = \frac{1}{120}\sum_{j=0}^{n_{m,n}-1} J_j, \tag{2.58}$$

where $n_{m,n}$ is the number of tetrahedra in the set $\mathcal{T}_{m,n}$, which is the intersection of sets $\mathcal{T}_m$ and $\mathcal{T}_n$. The linear polynomial $f(x,y,z)$ interpolating the values one, zero, zero, and zero at the vertices $(x_0,y_0,z_0)^{\mathrm{T}}$, $(x_1,y_1,z_1)^{\mathrm{T}}$, $(x_2,y_2,z_2)^{\mathrm{T}}$, and $(x_3,y_3,z_3)^{\mathrm{T}}$, respectively, has the partial derivatives

$$f^x(x,y,z) = -\frac{1}{J}\det\begin{bmatrix} 1 & y_1 & z_1\\ 1 & y_2 & z_2\\ 1 & y_3 & z_3\end{bmatrix}, \tag{2.59}$$

$$f^y(x,y,z) = -\frac{1}{J}\det\begin{bmatrix} x_1 & 1 & z_1\\ x_2 & 1 & z_2\\ x_2 & 1 & z_3\end{bmatrix}, \tag{2.60}$$

and

$$f^z(x,y,z) = -\frac{1}{J}\det\begin{bmatrix} x_1 & y_1 & 1\\ x_2 & y_2 & 1\\ x_3 & y_3 & 1\end{bmatrix}. \tag{2.61}$$

Integrals involving these partial derivatives are

$$\sum_{j=0}^{n_m-1}\int_{T_j}\Big(f_m^x(x,y,z)\Big)^2 dxdydz = \frac{1}{6}\sum_{j=0}^{n_m-1}\frac{1}{J_j}\left(\det\begin{bmatrix} 1 & y_1 & z_1\\ 1 & y_2 & z_2\\ 1 & y_3 & z_3\end{bmatrix}\right)^2, \tag{2.62}$$

$$\sum_{j=0}^{n_m-1} \int_{T_j} \left(f_m^y(x,y,z)\right)^2 dxdydz = \frac{1}{6} \sum_{j=0}^{n_m-1} \frac{1}{J_j} \left(\det \begin{bmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{bmatrix} \right)^2, \tag{2.63}$$

and

$$\sum_{j=0}^{n_m-1} \int_{T_j} \left(f_m^z(x,y,z)\right)^2 dxdydz = \frac{1}{6} \sum_{j=0}^{n_m-1} \frac{1}{J_j} \left(\det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \right)^2, \tag{2.64}$$

where the knots $(x_1^j, y_1^j, z_1^j)^{\mathrm{T}}$, $(x_2^j, y_2^j, z_2^j)^{\mathrm{T}}$, and $(x_3^j, y_3^j, z_3^j)^{\mathrm{T}}$ denote the boundary vertices of the faces of the platelet tetrahedra associated with knot $(x_m^j, y_m^j, z_m^j)^{\mathrm{T}}$. Other required values are

$$\sum_{j=0}^{n_{m,n}-1} \int_{T_j} f_m^x(x,y,z) f_n^x(x,y,z) dxdydz = \frac{1}{6} \sum_{j=0}^{n_{m,n}-1} \frac{1}{J_j} \det \begin{bmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{bmatrix} \det \begin{bmatrix} 1 & y_0 & z_0 \\ 1 & y_3 & z_3 \\ 1 & y_2 & z_2 \end{bmatrix}, \tag{2.65}$$

$$\sum_{j=0}^{n_{m,n}-1} \int_{T_j} f_m^y(x,y,z) f_n^y(x,y,z) dxdydz = \frac{1}{6} \sum_{j=0}^{n_{m,n}-1} \frac{1}{J_j} \det \begin{bmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{bmatrix} \det \begin{bmatrix} x_0 & 1 & z_0 \\ x_3 & 1 & z_3 \\ x_2 & 1 & z_2 \end{bmatrix}, \tag{2.66}$$

and

$$\sum_{j=0}^{n_{m,n}-1} \int_{T_j} f_m^z(x,y,z) f_n^z(x,y,z) dxdydz = \frac{1}{6} \sum_{j=0}^{n_{m,n}-1} \frac{1}{J_j} \det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \det \begin{bmatrix} x_0 & y_0 & 1 \\ x_3 & y_3 & 1 \\ x_2 & y_2 & 1 \end{bmatrix}, \tag{2.67}$$

where $n_{m,n}$ is the number of platelet tetrahedra in the set $\mathcal{T}_{m,n}$—the common platelet tetrahedra between knots $\mathbf{k}_m$ and $\mathbf{k}_n$—and $(x_0^j, y_0^j, z_0^j)^{\mathrm{T}}$, $(x_1^j, y_1^j, z_1^j)^{\mathrm{T}}$, $(x_2^j, y_2^j, z_2^j)^{\mathrm{T}}$, and $(x_3^j, y_3^j, z_3^j)^{\mathrm{T}}$ are vertices of a tetrahedron $T_j \in \mathcal{T}_{m,n}$.

Chapter 3

Linear-edge Quadratic Elements

Higher-order elements have gained in importance, since they can be used to represent complex data both in the context of numerical simulation and numerical data approximation. Figure 3.1 shows the advantage of using a higher-order element to approximate data in the 1D case. Higher-order elements can typically represent data better when compared to lower-order elements. This improvement in quality is true for two, three, and higher dimensions. In the 2D case, a *linear triangular* element represents a linear polynomial defined over the domain of the triangle. Most visualization and approximation techniques can use this type of element. A higher-order triangular element is the *quadratic triangle*, which has a quadratic polynomial defined over the same domain as the linear triangle. Figure 3.2 shows an example of a linear triangle and a quadratic triangle. In the 3D case, *linear tetrahedra* can be extended to *quadratic tetrahedra* in a similar fashion.

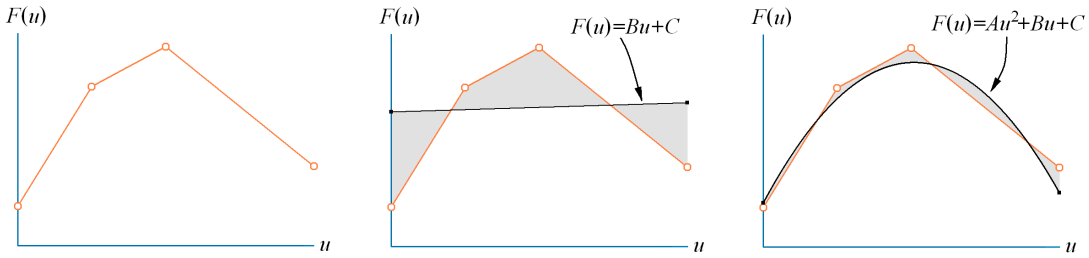

Figure 3.1: Advantage of using higher-order representation. Left image shows original piecewise linear data. Middle image shows linear approximation using one linear element. Right image shows quadratic approximation using one quadratic element. Gray area represents approximation error.

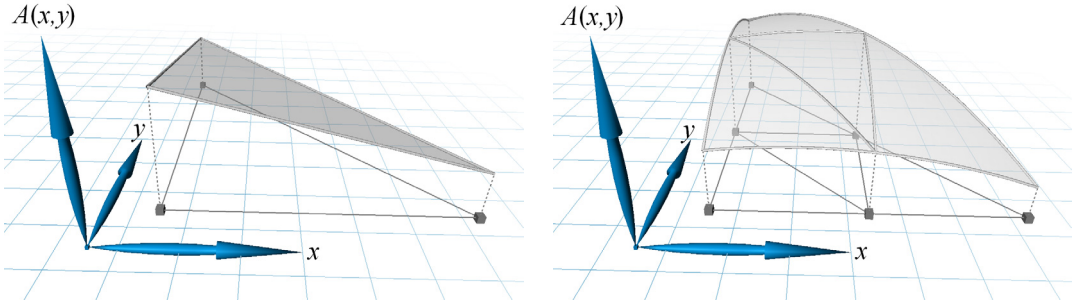

Figure 3.2: Left image shows a linear triangular element. Right image shows a quadratic triangular element.[1]

Higher-order hexahedral elements are popular in finite element applications [9], and the method described in [47] shows the potential for substantial reductions in the number of required elements when replacing linear elements with quadratic elements. The overall goal is the construction of a hierarchical data approximation over 2D and 3D domains using a best-approximation approach based on quadratic polynomials.

3.1 The 2D Case

Each simplicial element has six associated knots, one knot per corner, and one knot per edge. For simplicity, only edge knots that are positioned at the midpoint along the edges of the standard simplex are considered. A quadratic polynomial is associated with each simplicial element that approximates the dependent variable over the corresponding region in space. Each quadratic basis polynomial is represented in Bernstein-Bézier form, see [12]. Assuming that the function being approximated—typically a scalar- or vector-valued function—is known in analytical form, it is possible to compute the unique best quadratic spline approximation defined as a linear combination of a set of quadratic basis functions. The best approximation, understood in a least squares sense, is the result of solving the normal equations, see [10].

The standard triangle $T^{\mathbb{U}}$ in parameter space is the triangle with corners $(0,0)^{\mathrm{T}}$, $(1,0)^{\mathrm{T}}$, and $(0,1)^{\mathrm{T}}$. A 2D quadratic Bernstein-Bézier polynomial $B^2_{i,j}(u,v)$ (abbreviated

[1] In this dissertation, quadratic elements are often rendered with parametric lines in the interior of the element. These lines help show the interior curvature of the element.

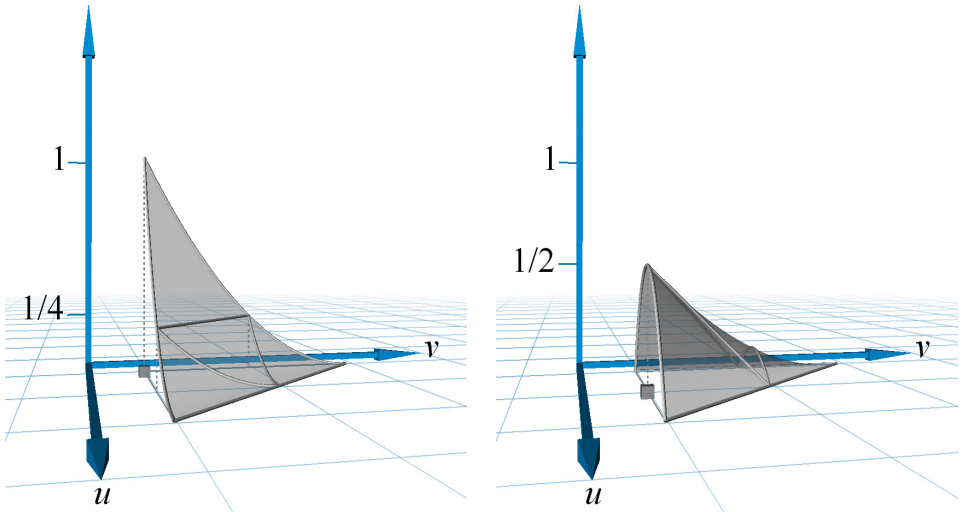

Figure 3.3: Bivariate quadratic basis (or "shape") functions. Left image shows the basis function $B^2_{0,0}(u,v) = (1-u-v)^2$ associated with corner knot located at $\mathbf{u}_{0,0} = (0,0)^{\mathrm{T}}$. Right image shows the basis function $B^2_{1,0}(u,v) = 2(1-u-v)u$ associated with edge knot located at $\mathbf{u}_{1,0} = (\frac{1}{2},0)^{\mathrm{T}}$.

$B^2_{i,j}$), is defined as

$$B^2_{i,j}(u,v) = \frac{2!}{(2-i-j)!i!j!}(1-u-v)^{2-i-j}u^i v^j,\ i,j \geq 0,\ i+j \leq 2, \tag{3.1}$$

and is associated with each corner and midpoint of each edge. The six basis polynomials correspond to the six knots $\mathbf{u}_{i,j} = (u_{i,j}, v_{i,j})^{\mathrm{T}} = \left(\frac{i}{2}, \frac{j}{2}\right),\ i,j \geq 0,\ i+j \leq 2$, in the standard triangle $T^{\mathbb{U}}$.

The analytical function being approximated is denoted by $F(x,y)$ (abbreviated as F). The normal equations determine the set of coefficients for the desired quadratic spline representation—a best approximation in the least squares sense.

Corner knots of simplicial elements may be shared by any number of elements, and the basis function associated with a corner knot $\mathbf{v}_i$ is denoted by $f_i(x,y)$. An edge of a simplicial element may be shared by no more than two elements in the 2D case and by an arbitrary number of elements in the 3D case. A basis function $g_j(x,y)$ is associated with the midpoint $\mathbf{e}_j$ of a simplex edge. Figure 3.3 shows the two types of basis functions for the 2D case. The set of elements sharing a common corner knot is called the platelet of this corner, and the set of elements sharing a common edge, are called "edge neighbors." Thus,

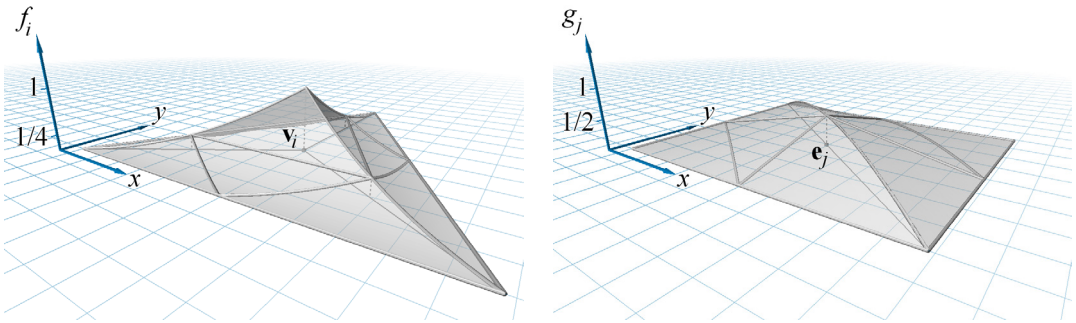

Figure 3.4: Basis functions associated with the platelet of knot $\mathbf{v}_i$ and the edge neighbors of edge $\mathbf{e}_j$.

a set of platelet elements defines the region in space over which a basis function associated with the corresponding corner knot is non-zero. Edge neighbors define the region in space over which a basis function, associated with this edge, is non-zero. Figure 3.4 shows the basis functions associated with the platelet of a vertex and the edge neighbors of an edge.

The best approximation $A(x, y)$ of a function $F(x, y)$ is defined as a linear combination of the basis functions associated with all distinct element corners ("corner basis functions" f_i) and simplex edges ("edge basis functions" g_i). Assuming that there are m distinct corners and n distinct edges, the best approximation is given as

$$A(x,y) = \sum_{i=0}^{m-1} c_i f_i(x,y) + \sum_{j=0}^{n-1} d_j g_j(x,y). \tag{3.2}$$

The normal equations are solved to obtain the unknown coefficients c_i and d_j. In matrix form, the normal equations are

$$\begin{bmatrix} \langle f_0, f_0\rangle & \cdots & \langle f_0, f_{m-1}\rangle & \langle f_0, g_0\rangle & \cdots & \langle f_0, g_{n-1}\rangle \\ \vdots & & & & & \vdots \\ \langle f_{m-1}, f_0\rangle & \cdots & \langle f_{m-1}, f_{m-1}\rangle & \langle f_{m-1}, g_0\rangle & \cdots & \langle f_{m-1}, g_{n-1}\rangle \\ \langle g_0, f_0\rangle & \cdots & \langle g_0, f_{m-1}\rangle & \langle g_0, g_0\rangle & \cdots & \langle g_0, g_{n-1}\rangle \\ \vdots & & & & & \vdots \\ \langle g_{n-1}, f_0\rangle & \cdots & \langle g_{n-1}, f_{m-1}\rangle & \langle g_{n-1}, g_0\rangle & \cdots & \langle g_{n-1}, g_{n-1}\rangle \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_{m-1} \\ d_0 \\ \vdots \\ d_{n-1} \end{bmatrix} = \begin{bmatrix} \langle F, f_0\rangle \\ \vdots \\ \langle F, f_{m-1}\rangle \\ \langle F, g_0\rangle \\ \vdots \\ \langle F, g_{n-1}\rangle \end{bmatrix}, \tag{3.3}$$

abbreviated as $M\mathbf{c} = \mathbf{F}$, where the inner product $\langle g, h\rangle$ of two functions $g(x, y)$ and $h(x, y)$ is similarly defined to the 1D scalar product given by Equation (2.2).

As in the linear case, the change-of-variables theorem is used to integrate over

arbitrary triangles. The scalar products one must consider are

$$\langle N_{0,0}, N_{0,0} \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} (1-u-v)^4 \; dudv = \frac{1}{30}, \tag{3.4}$$

$$\langle N_{0,0}, N_{1,0} \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} 2(1-u-v)^3 u \; dudv = \frac{1}{60}, \tag{3.5}$$

$$\langle N_{0,0}, N_{2,0} \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} (1-u-v)^2 u^2 \; dudv = \frac{1}{180}, \tag{3.6}$$

$$\langle N_{0,0}, N_{1,1} \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} 2(1-u-v)^2 uv \; dudv = \frac{1}{180}, \tag{3.7}$$

$$\langle N_{1,0}, N_{1,0} \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} 4(1-u-v)^2 u^2 \; dudv = \frac{1}{45}, \text{ and} \tag{3.8}$$

$$\langle N_{1,0}, N_{0,1} \rangle = \int_{v=0}^{1} \int_{u=0}^{1-v} 4(1-u-v)^2 uv \; dudv = \frac{1}{90}. \tag{3.9}$$

where $N_{i,j}$ corresponds to the basis function located at $\mathbf{u}_{i,j} = \left(\frac{i}{2}, \frac{j}{2}\right)^{\mathrm{T}}$.

The same Jacobian as in the linear case, see Equation (2.16), is used here, since the domain of the quadratic triangle has linearly defined edges. The global error E for approximation $A(x)$ and local errors e_i for an element i are computed using the same method as the linear case, using Equations (2.21) and (2.22), respectively. The linear system $M\mathbf{c} = \mathbf{F}$ is sparsely defined. An efficient sparse system solver is used to find the coefficients for the best approximation.

An example showing the potential element reduction by using quadratic elements is shown in Figure 3.5. The function being approximated is $F(x, y) = x^2 + y^2, \; -\frac{1}{2} \leq x, y \leq \frac{1}{2}$. In this ideal example (the original function is quadratic), a quadratic approximation having two elements can, in theory, represent this function exactly (numerical floating-point error is introduced in practice). To represent this function accurately, a linear approximation must use a relatively large number of elements—nearly 200 elements. (This implies an (approximate) upper bound to the potential element reduction—when comparing linear and quadratic elements in an ideal situation—of about one hundredth the number of linear elements.) The global error for the quadratic representation is 3.6×10^{-14} and is 1.6×10^{-6} for the linear approximation. The linear approximation was computed using the method described in Section 2.2.

A comparison of a quadratic- and a linear-spline approximation is shown in Figure 3.6. The original image consists of 1536×1024 RGB pixels (the described approximation

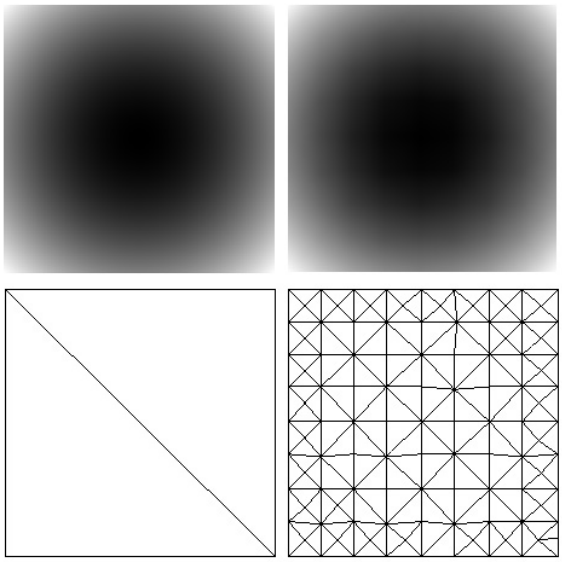

Figure 3.5: Comparison between quadratic spline representation (left) and linear spline approximation (right). The function being approximated is $F(x, y) = x^2 + y^2,\ -\frac{1}{2} \leq x, y \leq \frac{1}{2}$. Below each approximation is the corresponding domain decomposition. The quadratic representation uses 9 knots and two elements. The linear approximation uses 111 knots and 187 elements.

method was extended to support vector-valued data, thus, computing each channel of the RGB data simultaneously). The quadratic spline approximation consists of 2989 quadratic elements and the linear approximation uses 11482 linear elements. Computation time was 158 seconds for the quadratic approximation and 536 seconds for the linear approximation and image-space errors are 3.61% and 3.83%, respectively. Image-space errors are computed for the approximations by computing a difference image and then integrating over the result, see the Appendix, Section B.

A hierarchy of approximations can be constructed, as in the linear case, by computing element-specific errors e_i, refining the highest error elements, and recomputing the spline coefficients. A hierarchy of 2D quadratic spline approximations is shown in Figure 3.7. The original image consists of 211×144 pixels. Global errors for the four approximations are 37.05, 9.70, 1.86, and 0.45. Image-space errors for the four approximations are 8.44%, 6.95%, 5.90%, and 5.27%. Computation times ranged from six to 200 seconds for the four approximations. [2]

[2] All of the approximations were computed on a 1.8GHz Pentium IV graphics workstation with 512MB of main memory. Linear approximations were rendered at interactive frame rates. Quadratic approximations required just a few seconds to render (in software) per frame.

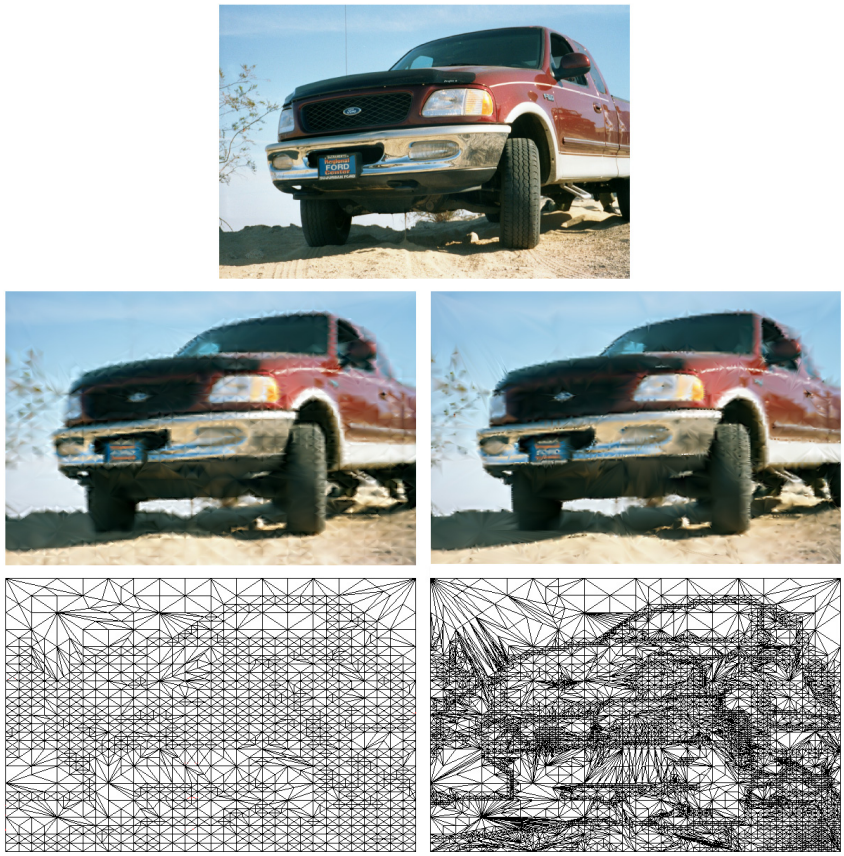

Figure 3.6: Comparison between quadratic spline approximation (left) and linear spline approximation (right). Original image is shown at the top. The quadratic approximation uses 6076 knots and 2989 elements. The linear approximation uses 5816 knots and 11482 elements. Image-space errors are 3.61% and 3.83% for the quadratic and linear approximation, respectively.

3.2 The 3D Case

The 3D case is a straightforward extension of the 2D case. Only the significant differences are described. The quadratic tetrahedron is defined by ten knots—four corner knots and six edge knots. The standard tetrahedron $T^{\mathbb{U}}$ in parameter space is the tetrahedron with corners $(0,0,0)^{\mathrm{T}}$, $(1,0,0)^{\mathrm{T}}$, $(0,1,0)^{\mathrm{T}}$, and $(0,0,1)^{\mathrm{T}}$. A 3D quadratic Bernstein-Bézier polynomial $B^2_{i,j,k}(u,v,w)$ (abbreviated $B^2_{i,j,k}$), is defined as

$$B^2_{i,j,k}(u,v,w) = \frac{2!}{(2-i-j-k)!i!j!k!}(1-u-v-w)^{2-i-j-k}u^i v^j w^k,$$
$$i,j,k \geq 0,\ i+j+k \leq 2, \tag{3.10}$$

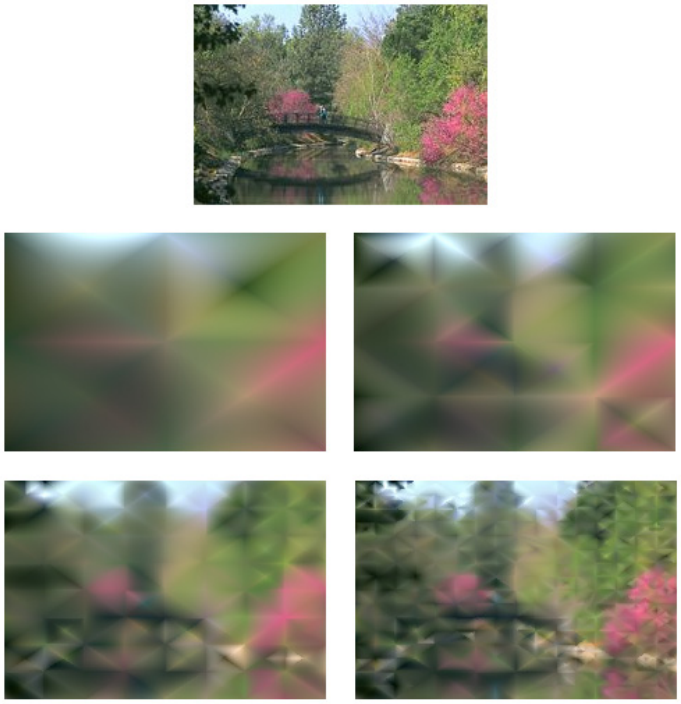

Figure 3.7: Quadratic hierarchical approximation of digital image data set. Original image is shown at the top. Four approximations are shown, from upper-left to lower-right, using 16, 48, 191, and 790 quadratic elements, respectively.

and is associated with each corner and midpoint of each edge. The ten basis polynomials correspond to the ten knots $\mathbf{u}_{i,j,k} = (u_{i,j,k}, v_{i,j,k}, w_{i,j,k})^{\mathrm{T}} = \left(\frac{i}{2}, \frac{j}{2}, \frac{k}{2}\right)^{\mathrm{T}}$, $i, j, k \geq 0$, $i+j+k \leq 2$ in parameter space. The scalar products one must consider are

$$\begin{aligned}
\langle N_{0,0,0}, N_{0,0,0} \rangle &= \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} (1-u-v-w)^4 \; dudv = \frac{1}{210}, \\
\langle N_{0,0,0}, N_{1,0,0} \rangle &= \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} 2(1-u-v-w)^3 u \; dudv = \frac{1}{420}, \\
\langle N_{0,0,0}, N_{2,0,0} \rangle &= \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} (1-u-v-w)^2 u^2 \; dudv = \frac{1}{1260}, \\
\langle N_{0,0,0}, N_{1,1,0} \rangle &= \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} 2(1-u-v-w)^2 uv \; dudv = \frac{1}{1260}, \\
\langle N_{1,0,0}, N_{1,0,0} \rangle &= \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} 4(1-u-v-w)^2 u^2 \; dudv = \frac{1}{315}, \\
\langle N_{1,0,0}, N_{0,1,0} \rangle &= \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} 4(1-u-v-w)^2 uv \; dudv = \frac{1}{630}, \text{ and} \\
\langle N_{1,0,0}, N_{0,1,1} \rangle &= \int_{w=0}^{1} \int_{v=0}^{1-w} \int_{u=0}^{1-v-w} 4(1-u-v-w) uvw \; dudv = \frac{1}{1260}, \qquad (3.11)
\end{aligned}$$

where $N_{i,j,k}$ corresponds to the basis function located at $\mathbf{u}_{i,j,k} = \left(\frac{i}{2}, \frac{j}{2}, \frac{k}{2}\right)^{\mathrm{T}}$. The same Jacobian as in the linear case, see Equation (2.25), is used here, since the mapping from the standard tetrahedron $T^{\mathbb{U}}$ to the linear-edge quadratic tetrahedron T is linear.

A comparison of a quadratic- and a linear-spline approximation of a 3D skull data set is shown in Figure 3.8. The original data set consists of 278528 data sites. The quadratic-spline approximation is visualized by tessellating each quadratic element with 512 linear elements and extracting an isosurface from the linear elements. The isosurface for the linear-spline approximation was extracted directly from the linear elements forming the approximation. The quadratic-spline approximation has a global error of 2.15×10^{-6}, and the linear spline approximation has a global error of 1.65×10^{-2}. Image-space errors are 8.12% and 8.33% for the isosurface images from the quadratic and linear approximation, respectively, see the Appendix Section B. The quadratic-spline approximation required about 20 hours of computation time while the linear-spline approximation required less than three.

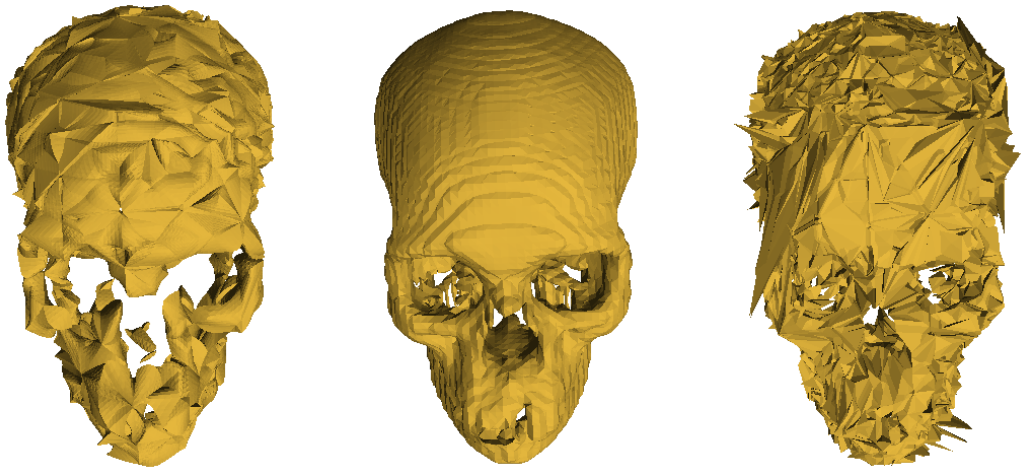

Figure 3.8: Comparison between quadratic approximation (left) and linear approximation (right). Original skull is shown in the center image. The quadratic approximation uses 7487 knots and 5348 elements. The linear approximation uses 14667 knots and 78530 elements. Image-space errors are 8.12% and 8.33% for the isosurface images from the quadratic and linear approximation, respectively.

A hierarchy of approximations can be constructed, as in the linear case, by computing element-specific errors e_i, refining the highest error elements, and recomputing the spline coefficients. A sample hierarchy of 3D quadratic spline approximations for a 3D skull

data set is shown in Figure 3.9. Global errors for the four approximations are 1.0×10^{-3}, 4.7×10^{-4}, 3.9×10^{-5}, and 2.1×10^{-6}. [3]

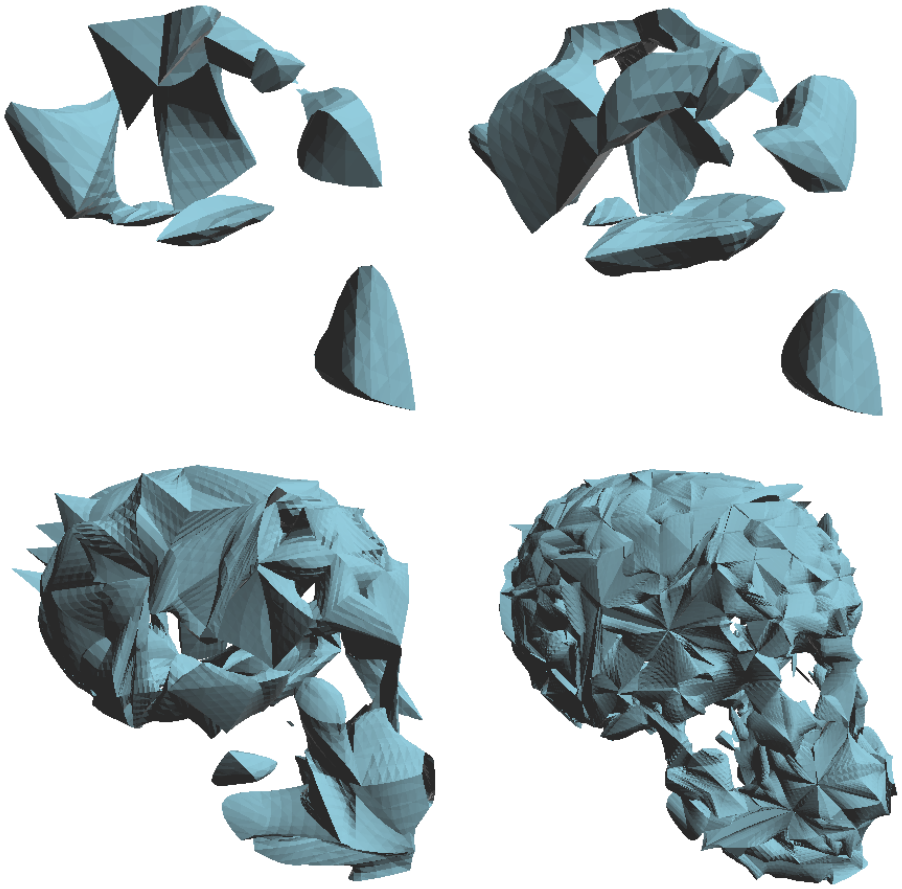

Figure 3.9: Quadratic hierarchical approximation of skull data set. Four approximations, from upper-left to lower-right, using 62, 125, 741, and 5384 quadratic elements, respectively.

[3] All of the approximations were computed on a 1.8GHz Pentium IV graphics workstation with 512MB of main memory. Linear approximations were rendered at interactive frame rates. Tessellation of quadratic approximations required several seconds. Once tessellated, computing and rendering an isosurface was performed at interactive frame rates.

Chapter 4

Curved-quadratic Elements

The *linear-edge higher-order* elements described in Chapter 3 are extended to include higher-order ("curved") domains. These elements are called *curved higher-order* elements or, more simply, *curved elements*. A *curved-quadratic* triangle (and tetrahedron) is defined as an element that has both a quadratically defined domain and quadratic polynomial defined over that domain, see Figure 4.1.

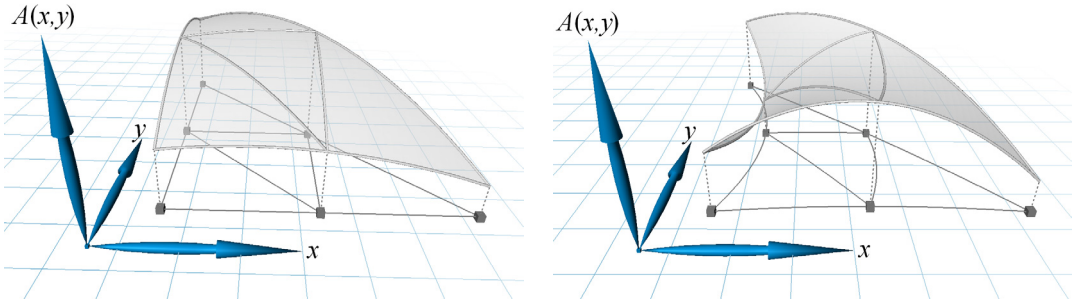

Figure 4.1: Comparison of linear-edge quadratic element to curved quadratic element. Left image shows a linear-edge quadratic triangular element. Right image shows a curved-quadratic triangular element. Curved-quadratic element is curved in both functional space and xy-space.[1]

Curved-quadratic elements can be used to decompose a domain more effectively. For example, consider a cross section of a wing and the air flow around it. There are distinct regions, defined by discontinuities in the data, that can be better represented by aligning

[1]In this dissertation, quadratic elements are often rendered with parametric lines in the interior of the element. These lines help show the interior curvature of the element.

element edges along these discontinuities, see Figure 4.2.

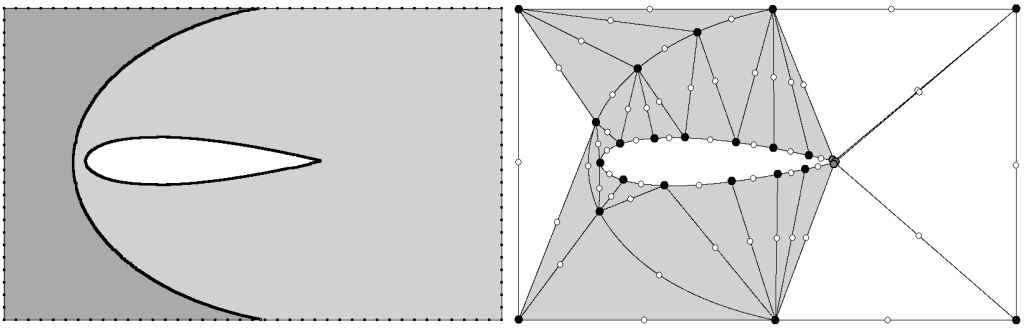

Figure 4.2: Decomposition of domain around a wing using 2D curved-quadratic elements. Left image shows three distinct regions of interest, each separated by discontinuities. Dark gray and light gray regions are on either side of a "shock." White region is region bounded by wing geometry. Right image shows possible domain decomposition using a combination of curved-quadratic (gray) and linear-edge-quadratic (white) elements. Bullets denote corner vertices; circles denote edge vertices.

4.1 The 2D Case

Computation of the best approximation is very similar to that of the linear-edge quadratic element. The only difference is in the implementation of the change-of-variables theorem. In this case, the Jacobian $J(u,v)$ cannot be moved outside of the integral computations, since it depends on u and v. The Jacobian for a curved-quadratic triangle is given as

$$J(u,v) = \det \begin{bmatrix} \frac{\partial}{\partial u}x(u,v) & \frac{\partial}{\partial v}x(u,v) \\ \frac{\partial}{\partial u}y(u,v) & \frac{\partial}{\partial v}y(u,v) \end{bmatrix}, \tag{4.1}$$

where the partial derivatives are

$$\begin{aligned} \frac{\partial}{\partial u}x(u,v) &= 2((x_5 - x_4 - x_3 + x_0)v + (x_1 - 2x_3 + x_0)u + x_3 - x_0), \\ \frac{\partial}{\partial v}x(u,v) &= 2((x_2 - 2x_4 + x_0)v + (x_5 - x_4 - x_3 + x_0)u + x_4 - x_0), \\ \frac{\partial}{\partial u}y(u,v) &= 2((y_5 - y_4 - y_3 + y_0)v + (y_1 - 2y_3 + y_0)u + y_3 - y_0), \text{ and} \\ \frac{\partial}{\partial v}y(u,v) &= 2((y_2 - 2y_4 + y_0)v + (y_5 - y_4 - y_3 + y_0)u + y_4 - y_0), \end{aligned} \tag{4.2}$$

where the six knots $\mathbf{k}_i = (x_i, y_i)^{\mathrm{T}}$ of the element are ordered as shown in Figure 4.3.

A curved-quadratic triangle approximation is applied to the 2D digital image data shown in Figure 4.4. This image has three regions that are distinctly separated by disconti-

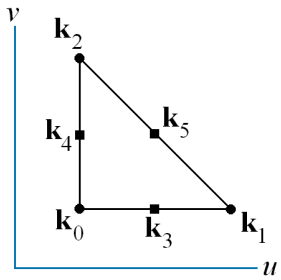

Figure 4.3: Indexing used for curved-quadratic triangle.

Figure 4.4: Left image shows original digital image that can benefit from curved-quadratic element approximation. Middle image shows a curved-quadratic approximation consisting of 411 knots and 134 curved-quadratic triangles. Right image shows the triangulation of the knots. Approximation has an image-space error of 1.14%.

nuities. An approximation using curved-quadratic triangles was constructed by first fitting quadratic curves to the discontinuities and then triangulating the result. Since the three different regions are separated by discontinuities, a separate approximation was computed for each region. Knots along the discontinuities may share the same location, but have different coefficient values associated with them.

A comparison between a linear and a curved-quadratic triangle approximation is shown in Figure 4.5. A hierarchy of curved-quadratic approximations was constructed and then two levels in the hierarchy were extracted for comparison. Each of the three regions were treated independently. Thus, there are "hanging nodes" along the discontinuity. However, this is acceptable in this situation because the knots lie along the discontinuity. Image-space errors for the low-resolution quadratic, high-resolution quadratic, and linear approximation are 1.14%, 0.63%, and 1.77%, respectively, see the Appendix, Section B.

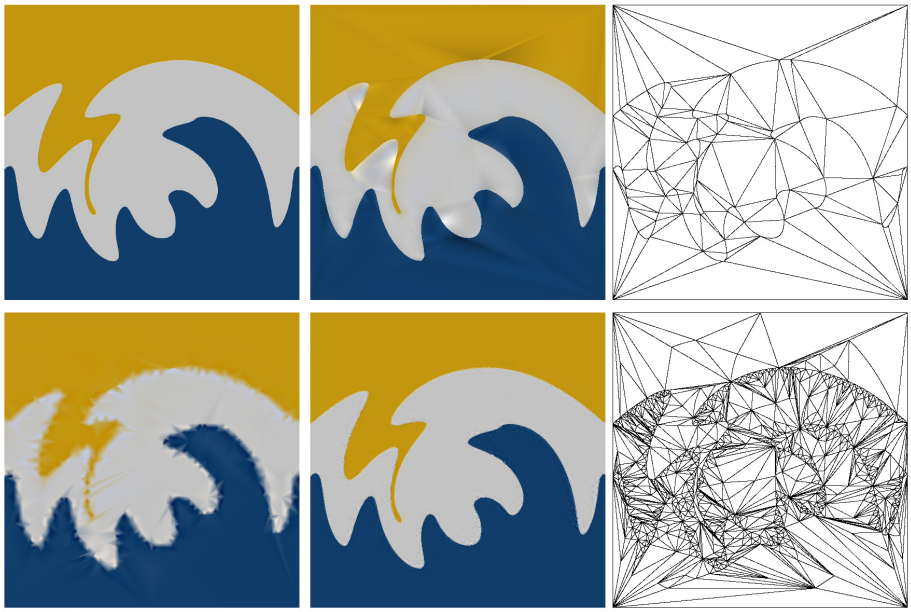

Figure 4.5: Comparison between curved-quadratic and linear approximations. Original image, consisting of 800×800 pixels, is shown in the upper-left corner. One linear approximation is shown in the lower-left corner. Two curved-quadratic approximations are shown in the middle column; their corresponding triangulations are shown on the right side. The linear approximation has 754 knots, 1483 linear simplices, an image-space error of 1.77%, and a computation time of 42 seconds. The upper quadratic approximation has 411 knots, 134 quadratic simplices, an image-space error of 1.14%, and a computation time of 24 seconds. The bottom quadratic approximation has 3392 knots, 1473 quadratic simplices, an image-space error of 0.63%, and a computation time of 82 seconds.

Refining a curved-quadratic triangle

Bisection of curved-quadratic triangles is a bit more complicated than bisection of linear-edge triangles. A triangle T—defined by six knots $\mathbf{k}_i$, $0 \leq i \leq 5$—is bisected (along the edge connecting $\mathbf{k}_0$, $\mathbf{k}_3$, and $\mathbf{k}_1$) into two triangles T_a and T_b according to the new knot locations $\mathbf{a}_i$ and $\mathbf{b}_i$, as shown in Figure 4.6, given as

$$\begin{aligned}
\mathbf{a}_0 &= \mathbf{k}_0, \\
\mathbf{a}_1 = \mathbf{b}_0 &= \frac{\mathbf{k}_0 + 2\mathbf{k}_3 + \mathbf{k}_1}{4}, \\
\mathbf{a}_2 = \mathbf{b}_2 &= \mathbf{k}_2, \\
\mathbf{a}_3 &= \frac{\mathbf{k}_0 + \mathbf{k}_3}{2}, \\
\mathbf{a}_4 &= \mathbf{k}_4, \\
\mathbf{a}_5 = \mathbf{b}_4 &= \frac{\mathbf{k}_4 + \mathbf{k}_5}{2},
\end{aligned}$$

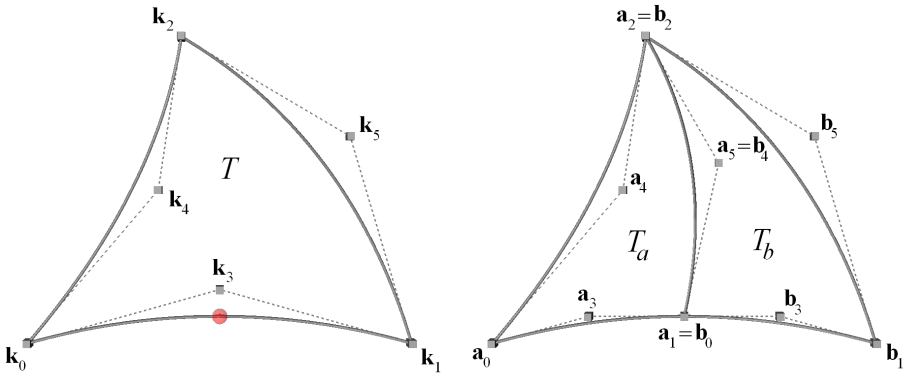

Figure 4.6: Bisection of a curved-quadratic triangle T. Left image shows triangle being bisected. Red dot indicates bisection point along edge connecting $\mathbf{k}_0$, $\mathbf{k}_3$, and $\mathbf{k}_1$. Right image shows new triangles T_a and T_b resulting from bisection.

$$\begin{aligned} \mathbf{b}_1 &= \mathbf{k}_1, \\ \mathbf{b}_3 &= \frac{\mathbf{k}_1 + \mathbf{k}_3}{2}, \text{ and} \\ \mathbf{b}_5 &= \mathbf{k}_5. \end{aligned} \tag{4.3}$$

4.2 The 3D Case

The 3D case is a straightforward extension of the 2D case. The only difference is in the implementation of the change-of-variables theorem. The Jacobian for a curved-quadratic tetrahedron is given as

$$J(u,v,w) = \det \begin{bmatrix} \frac{\partial}{\partial u}x(u,v,w) & \frac{\partial}{\partial v}x(u,v,w) & \frac{\partial}{\partial w}x(u,v,w) \\ \frac{\partial}{\partial u}y(u,v,w) & \frac{\partial}{\partial v}y(u,v,w) & \frac{\partial}{\partial w}y(u,v,w) \\ \frac{\partial}{\partial u}z(u,v,w) & \frac{\partial}{\partial v}z(u,v,w) & \frac{\partial}{\partial w}z(u,v,w) \end{bmatrix}, \tag{4.4}$$

where the partial derivatives are

$$\begin{aligned} \frac{\partial}{\partial u}x(u,v,w) &= 2((x_7 - x_4 + x_0 - x_5)v + (x_1 + x_0 - 2x_4)u + \\ &\quad (x_8 + x_0 - x_6 - x_4)w + x_4 - x_0), \\ \frac{\partial}{\partial v}x(u,v,w) &= 2((x_2 - 2x_5 + x_0)v + (x_7 - x_4 + x_0 - x_5)u + \\ &\quad (x_0 + x_9 - x_5 - x_6)w + x_5 - x_0), \end{aligned}$$

$$
\begin{aligned}
\frac{\partial}{\partial w}x(u,v,w) &= 2((x_0+x_9-x_5-x_6)v+(x_8+x_0-x_6-x_4)u+ \\
&\quad (x_0+x_3-2x_6)w+x_6-x_0), \\
\frac{\partial}{\partial u}y(u,v,w) &= 2((y_7-y_4+y_0-y_5)v+(y_1+y_0-2y_4)u+ \\
&\quad (y_8+y_0-y_6-y_4)w+y_4-y_0), \\
\frac{\partial}{\partial v}y(u,v,w) &= 2((y_2-2y_5+y_0)v+(y_7-y_4+y_0-y_5)u+ \\
&\quad (y_0+y_9-y_5-y_6)w+y_5-y_0), \\
\frac{\partial}{\partial w}y(u,v,w) &= 2((y_0+y_9-y_5-y_6)v+(y_8+y_0-y_6-y_4)u+ \\
&\quad (y_0+y_3-2y_6)w+y_6-y_0), \\
\frac{\partial}{\partial u}z(u,v,w) &= 2((z_7-z_4+z_0-z_5)v+(z_1+z_0-2z_4)u+ \\
&\quad (z_8+z_0-z_6-z_4)w+z_4-z_0), \\
\frac{\partial}{\partial v}z(u,v,w) &= 2((z_2-2z_5+z_0)v+(z_7-z_4+z_0-z_5)u+ \\
&\quad (z_0+z_9-z_5-z_6)w+z_5-z_0), \text{ and} \\
\frac{\partial}{\partial w}z(u,v,w) &= 2((z_0+z_9-z_5-z_6)v+(z_8+z_0-z_6-z_4)u+ \\
&\quad (z_0+z_3-2z_6)w+z_6-z_0), \qquad (4.5)
\end{aligned}
$$

where the ten knots $\mathbf{k}_i = (x_i, y_i, z_i)^{\mathrm{T}}$ of the element are ordered as shown in Figure 4.7.

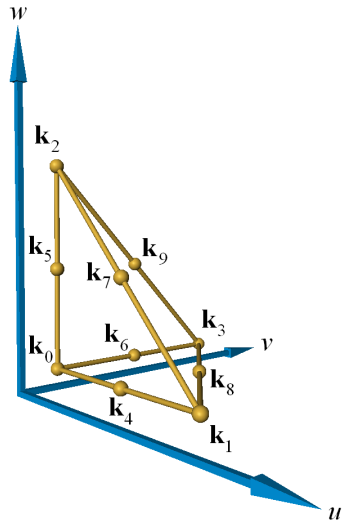

Figure 4.7: Indexing used for curved-quadratic tetrahedron.

Refining a curved-quadratic tetrahedron

Bisection of curved-quadratic tetrahedra is a bit more complicated than bisection of linear-edge tetrahedra. A tetrahedron T—defined by ten knots $\mathbf{k}_i,\ 0 \leq i \leq 9$—is bisected (along the edge connecting $\mathbf{k}_0$, $\mathbf{k}_4$, and $\mathbf{k}_1$) into two tetrahedra T_a and T_b according to the new knot locations $\mathbf{a}_i$ and $\mathbf{b}_i$, as shown in Figure 4.8, given as

$$\begin{aligned}
\mathbf{a}_0 &= \mathbf{k}_0, \\
\mathbf{a}_1 = \mathbf{b}_0 &= \frac{\mathbf{k}_0 + 2\mathbf{k}_4 + \mathbf{k}_1}{4}, \\
\mathbf{a}_2 = \mathbf{b}_2 &= \mathbf{k}_2, \\
\mathbf{a}_3 = \mathbf{b}_3 &= \mathbf{k}_3, \\
\mathbf{a}_4 &= \frac{\mathbf{k}_0 + \mathbf{k}_4}{2}, \\
\mathbf{a}_5 &= \mathbf{k}_5, \\
\mathbf{a}_6 &= \mathbf{k}_6, \\
\mathbf{a}_7 = \mathbf{b}_5 &= \frac{\mathbf{k}_5 + \mathbf{k}_7}{2}, \\
\mathbf{a}_8 = \mathbf{b}_6 &= \frac{\mathbf{k}_6 + \mathbf{k}_8}{2}, \\
\mathbf{a}_9 = \mathbf{b}_9 &= \mathbf{k}_9, \\
\mathbf{b}_1 &= \mathbf{k}_1, \\
\mathbf{b}_4 &= \frac{\mathbf{k}_1 + \mathbf{k}_4}{2}, \\
\mathbf{b}_7 &= \mathbf{k}_7, \text{ and} \\
\mathbf{b}_8 &= \mathbf{k}_8.
\end{aligned} \tag{4.6}$$

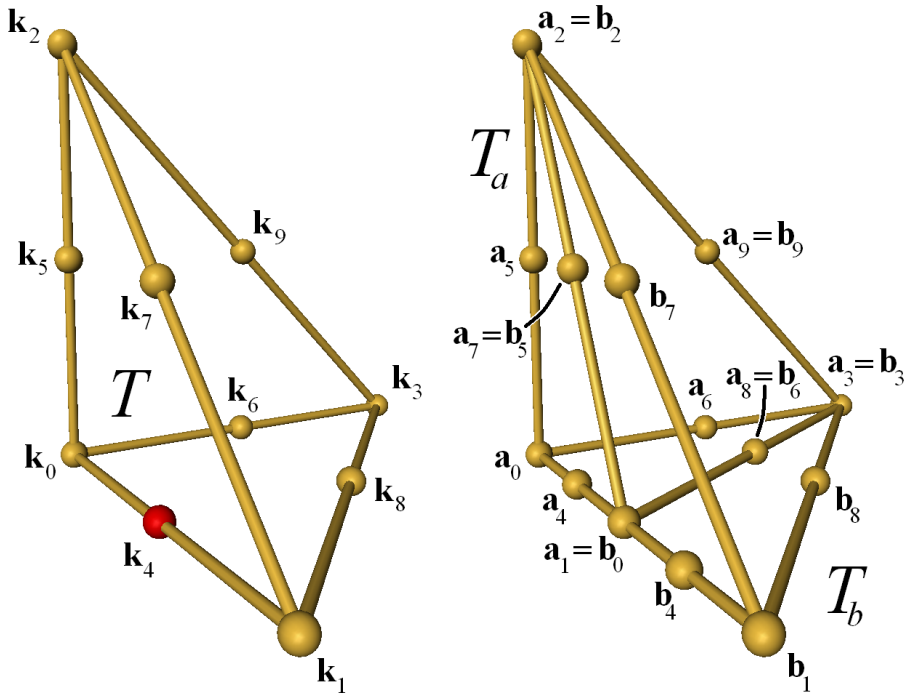

Figure 4.8: Bisection of a curved-quadratic tetrahedron T. Left image shows tetrahedron being bisected. Red dot indicates bisection point along edge connecting $\mathbf{k}_0$, $\mathbf{k}_4$, and $\mathbf{k}_1$. Right image shows new tetrahedra T_a and T_b resulting from bisection.

Part II

Visualization

Visualization of linear elements has been studied much more extensively than that for higher-order elements. Thus, it is not necessary to discuss the details of linear-element visualization. However, for higher-order methods to be competitive with linear elements, higher-order visualization techniques must be competitive.

If it is possible to replace several linear elements with higher-order ones, then the higher-order elements need to be visualized at least as fast as the linear elements they're replacing. However, considering the grander scheme of large scale visualization, higher-order elements may be able to borrow extra time from being transported more efficiently across networks—either LANs or video. If higher-order elements can reduce the required number of elements by 90%, even if it takes ten times longer to visualize a higher-order element over a linear element, higher-order elements still win, since they require less storage space and can be transported more efficiently.

Typically, higher-order elements are tessellated by several smaller linear elements for rendering purposes. Conventional visualization methods can be applied directly to these linear elements. Three fundamental visualization techniques including isosurfacing, ray casting, and cutting planes are discussed for quadratic and curved-quadratic elements. Each of the methods described strengthen the foundation of higher-order-element visualization.

Chapter 5

Isosurfacing (or Contouring)

An isosurface (or contour) c of a function $F(\mathbf{u})$ is all the values of $\mathbf{u}$ such that $F(\mathbf{u}) = c$. An isoline on an elevation map is an example of an isosurface in 2D. Each line represents a constant height over the map. Considering several lines at once, one can visually determine the "steepness" of the height field at any location by judging how close these lines are to each other. The most popular method for computing an isosurface in 3D is that of *marching cubes*, see [33]. This method works primarily on rectilinear gridded data, however, the idea can be generalized to tetrahedra, see [7]. In each of these methods, a divide-and-conquer approach is used to extract an isosurface from simplicial elements one at a time. These concepts are extended to higher-order elements.

A method for extracting a contour line (2D case) and an isosurface (3D case) from a higher-order element is shown—specifically from a quadratic triangle and tetrahedron. A method to transform the resulting contour line (or isosurface) into a quartic curve (or surface) based on a curved-triangle (curved-tetrahedron) mapping is shown. Figure 5.1 shows a 2D example of a contour line extracted from a curved-quadratic triangle. A contour through a 2D quadratic function defined over the standard triangle $T^{\mathbb{U}}$ in parameter space is a conic section and can be represented by a rational-quadratic function [50]. The rational-quadratic curve representing the contour is transformed into physical space based on a curved-quadratic triangle transformation, which forms a rational-quartic curve. An isosurface in the 3D case is approximated by a rational-quadratic patch over the standard

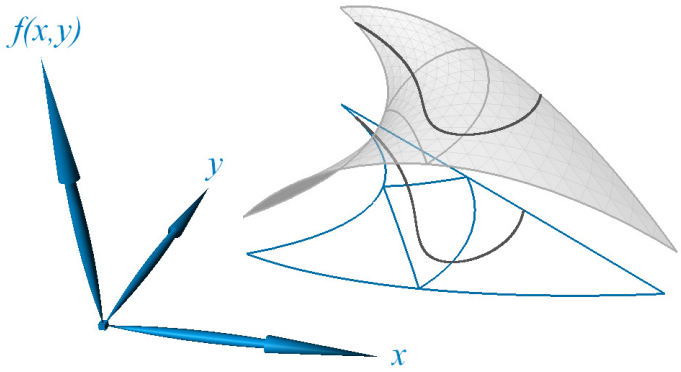

Figure 5.1: Contour of curved-quadratic triangle in physical space $\mathbb{R}^2$. Dark curves indicate contour in xy-plane (domain space) and on "graph" surface in 3D space.[1]

tetrahedron $T^{\mathbb{U}}$ in parameter space and is transformed to a rational-quartic patch in physical space based on a curved-tetrahedron transformation.

A curved-quadratic element is isosurfaced by first mapping the quadratic function $\mathbf{F}(\mathbf{u}) : \mathbb{U} \longrightarrow \mathbb{R}$ defined over the element into parameter space $\mathbb{U}$. Finding the representation $\mathbf{Q}(\mathbf{u}) : \mathbb{U} \longrightarrow \mathbb{U}$ for a contour value c such that $\mathbf{F}(\mathbf{Q}(\mathbf{u})) = c$, and then transforming $\mathbf{Q}(\mathbf{u})$ into physical space $\mathbb{R}$, yields the mapping $\mathbf{C}(\mathbf{u}) : \mathbb{U} \longrightarrow \mathbb{R}$—the representation of c in physical space. For both the 2D and 3D case, the transformation from a rational-quadratic function in $\mathbb{U}$ to rational-quartic functions in $\mathbb{R}$ for a curved-quadratic element transformation is shown. The resulting contour surfaces can be rendered efficiently in hardware.[2]

Finding contours (isosurfaces) in 2D and 3D linear-edge quadratic elements (i.e., parameter space) is discussed first. Then, the method to transform a parameter-space contour into the physical space defined by the curved-quadratic element is shown.

5.1 Previous Work

Few higher-order element visualization techniques exist. Higher-order hexahedra visualization is described in [31]. Visualization of higher-order element isosurfaces in the form

[1] In this dissertation, quadratic elements are often rendered with parametric lines in the interior of the element. These lines help show the interior curvature of the element.

[2] The ELSA Gladiac 920, nVidia GeForce 3 and GeForce 4, and ATI Radeon 8500 and Radeon 9700 [43] video hardware all support varying levels of higher-order patch rendering suitable for quartic patches.

of *A-patches* is described in [1]. Elements with a higher-order domain and a linearly defined polynomial defined over that domain are volumetrically visualized by the method in [35]. Creation of hierarchical quadratic-tetrahedral approximations is discussed in [47]. A quadratic tetrahedra volume renderer is described in [49].

Extracting isosurfaces from linear-edge quadratic triangles has been studied in [3, 34, 50]. The Worsey-Farin method [50] uses a Bernstein-Bézier basis, which tends to work better than the monomial basis used in the Marlow-Powell method, see [34]. The Worsey-Farin method and the method discussed by Bloomquist (in his thesis [3]) provide a foundation for finding contours in quadratic elements in their parameter spaces (i.e., linear-edge quadratic simplices). Bloomquist used the Worsey-Farin method for the 2D case and extended it to the 3D case to find contour surface intersections with the faces of a tetrahedron. The method described in the following sections is discussed in [46].

5.2 Linear-edge Quadratic Elements

5.2.1 The 2D Case

The Worsey-Farin method [50] is implemented to find rational-quadratic curves that represent the contour passing through a linear-edge quadratic triangle. The domain $\mathbb{U} \subseteq \mathbb{R}$ of the standard triangle $T^{\mathbb{U}}$—with vertices $(0,0)^{\mathrm{T}}$, $(1,0)^{\mathrm{T}}$, and $(0,1)^{\mathrm{T}}$—defines the *parameter space* $\mathbb{U}$ and the *physical space* $\mathbb{R}$. The contour in a quadratic triangle can be quite complex, and it is often desirable to represent it by several segments. (Consider the case when a contour is completely contained inside a triangle. In this case, three curves are used to represent the contour, see [50].) A univariate *rational-quadratic curve* $\mathbf{Q}(u) : \mathbb{U}^1 \longrightarrow \mathbb{U}^2$ that represents a segment of the contour, in Bernstein-Bézier form, with three control points $\mathbf{p}_i \in \mathbb{U}^2$ and three weights $w_i, 0 \leq i \leq 2$, $w_i \geq 0$, is defined as

$$\mathbf{Q}(u) \;=\; \frac{\sum_{i=0}^{2} w_i \mathbf{p}_i B_i^2(u)}{\sum_{i=0}^{2} w_i B_i^2(u)}, \tag{5.1}$$

where the univariate n^{th}-degree Bernstein polynomial $B_i^n(u)$ is

$$B_i^n(u) \;=\; \frac{n!}{(n-i)!i!}(1-u)^{n-i}u^i. \tag{5.2}$$

Worsey and Farin restrict the resulting rational-quadratic curve $\mathbf{Q}(u)$ such that $w_0 = 1$ and $w_2 = 1$, since a rational-quadratic curve, having w_0 and w_2 set to one, represents a conic section exactly, see [32]. Thus, the rational-quadratic curve used to represent a contour segment is defined as

$$\mathbf{Q}(u) \;=\; \frac{\mathbf{p}_0 B_0^2(u) + w_1 \mathbf{p}_1 B_1^2(u) + \mathbf{p}_2 B_2^2(u)}{B_0^2(u) + w_1 B_1^2(u) + B_2^2(u)}. \tag{5.3}$$

5.2.2 The 3D Case

A method similar to [24] is used to extend Bloomquist's 3D method by forming triangular rational-quadratic patches that represent the isosurface in a linear-edge quadratic tetrahedron. The rational-quadratic patch is formed by first applying the 2D method to each of the tetrahedron's faces to find the contour intersections. Then, patches are formed—from the contour lines on the faces—that approximate the isosurface. A triangular *rational-quadratic patch* $\mathbf{Q}(u,v) : \mathbb{U}^2 \longrightarrow \mathbb{U}^3$ that represents a region of the isosurface, in Bernstein-Bézier form, with six control points $\mathbf{p}_{ij} \in \mathbb{U}^3$ and six weights $w_{ij}, i, j \geq 0,\ i + j \leq 2,\ w_{ij} \geq 0$, is defined as

$$\mathbf{Q}(u,v) \;=\; \frac{\sum_{j=0}^{2} \sum_{i=0}^{2-j} w_{ij} \mathbf{p}_{ij} B_{ij}^2(u,v)}{\sum_{j=0}^{2} \sum_{i=0}^{2-j} w_{ij} B_{ij}^2(u,v)}, \tag{5.4}$$

where the bivariate n^{th}-degree Bernstein polynomial $B_{ij}^n(u,v)$ is

$$B_{ij}^n(u,v) \;=\; \frac{n!}{(n-i-j)!i!j!}(1-u-v)^{n-i-j} u^i v^j. \tag{5.5}$$

Patches representing the isosurface are constructed from rational-quadratic curves found on the faces of a tetrahedron. These rational-quadratic curves have their endpoint weights (w_0 and w_2) set to one, thus, when these curves are used to construct a rational-quadratic patch, the corner weights of the patch assume a weight of one. Thus, weights w_{00}, w_{20}, and w_{02} are all set to one, yielding the representation

$$\mathbf{Q}(u,v) \;=\; \frac{\begin{array}{c}\mathbf{p}_{00} B_{00}^2(u,v) + \mathbf{p}_{20} B_{20}^2(u,v) + \mathbf{p}_{02} B_{02}^2(u,v) + \\ w_{10}\mathbf{p}_{10} B_{10}^2(u,v) + w_{01}\mathbf{p}_{01} B_{01}^2(u,v) + w_{11}\mathbf{p}_{11} B_{11}^2(u,v)\end{array}}{\begin{array}{c} B_{00}^2(u,v) + B_{20}^2(u,v) + B_{02}^2(u,v) + \\ w_{10} B_{10}^2(u,v) + w_{01} B_{01}^2(u,v) + w_{11} B_{11}^2(u,v)\end{array}}. \tag{5.6}$$

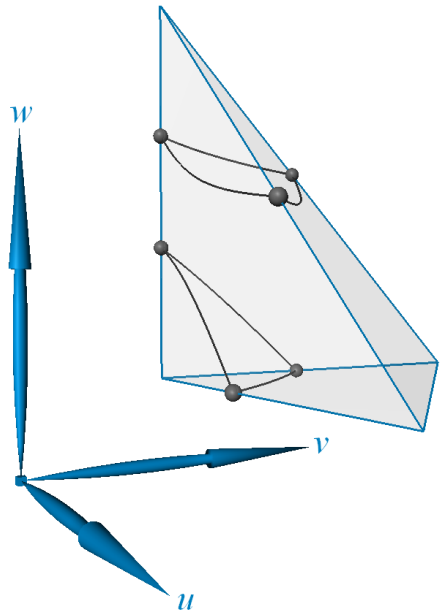

Figure 5.2: Two contour surfaces inside a quadratic tetrahedron. Dark dots are contour intersections with edges. Dark curves are face-intersections curves. There are two groups of three curves that bound two independent surfaces of contour.

Assembling isosurfaces

The 2D algorithm is applied to each face of a tetrahedron to find the intersections of the isosurface with each face; these intersections are called *face-intersection curves*. Since there can be more than one surface passing through a quadratic tetrahedron, the face-intersection curves are connected end-to-end to form groups of curves that bound various portions of the isosurface, see Figure 5.2. This method is similar to the method described in [21, 22]. Each group is classified according to the number of curves it contains:

- **No curve**. Either the contour surface is not present or the surface is "pill-shaped" and lies completely inside the tetrahedron.
- **One curve**. Only one edge of the tetrahedron is equal to the contour value, thus, it is not treated, i.e., the coefficients along that edge are equal to the contour value and all the others are either greater- or less-than the contour value.
- **Two curves**. Along one edge, the isosurface intersects two neighboring faces and looks similar to the "peel-of-an-orange slice," see Figure 5.3 (left).

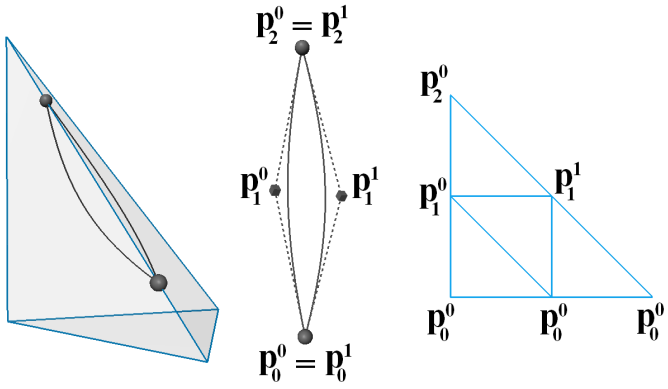

Figure 5.3: Constructing a triangular patch from two curves. One edge of patch is collapsed by using point $\mathbf{p}_0^0$ three times along an edge. Left image shows isosurface intersecting faces of tetrahedron. Middle image shows labelled points of two-curve boundary polygon. Right image shows patch indexing.

- **Three curves**. The surface intersects three neighboring faces or the surface exits through a face and does not intersect any edges defining that face.

- **More than three curves**. The surface is bounded by several curves.

Simple cases occur when there are two or three face-intersection curves bounding the surface. An approximation to the isosurface is found by representing the surface with one patch.

When there are two curves, one triangular patch is formed by collapsing one side of the patch to the same point, see Figure 5.3. A "crack" would be introduced if the surface were split across the middle to form two patches, since the curves found in neighboring elements would not necessarily be split. Later in the rendering process—when the patch is tessellated either in software or hardware—the degenerate patch edge produces zero-area triangles (where two vertices have the same location). In terms of visualization, no significant problems are introduced, since normal vectors for the vertices are computed analytically from the patch. Three curves are converted easily into one triangular patch by using the control points from the three boundary curves as patch control points.

More than three curves bounding the surface is non-trivial. Figure 5.4 shows an example of this type of complicated surface. First, a polygon is formed from the control nets of the face-intersection curves that bounds the surface; this polygon is always closed but

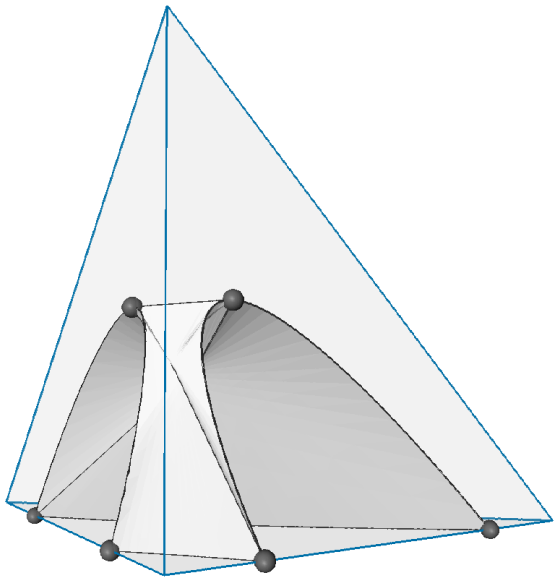

Figure 5.4: Contour surface bounded by six face-intersection curves. Dark dots are endpoints of face-intersection curves.

not necessarily convex. Three steps are performed to represent the surface with rational-quadratic patches:

1. Choose the shortest diagonal in the polygon to "split across." Here, a diagonal splits the polygon into two halves. Only diagonals that connect endpoints of the face-intersection curves are considered. If n is the number of boundary curves, then the only valid diagonals to choose from are those that partition the polygon into two sets of $\frac{n}{2}$ curves (or additionally $\frac{n+1}{2}$ when n is odd).

2. Choose a control point and weight for the center knot of the diagonal.

3. Recurse on each half until the simple case of three boundary curves is reached.

There are several possibilities to choose from when fixing the location of the center control point along a diagonal. Initially, this point was chosen by intersecting tangent planes of the isosurface. However, this method turned out to be inappropriate, since the intersection quite often lay outside the tetrahedron.

A more stable approach that considers various combinations of the center control points of the face-intersection curves ensures a point that lies inside the tetrahedron. For

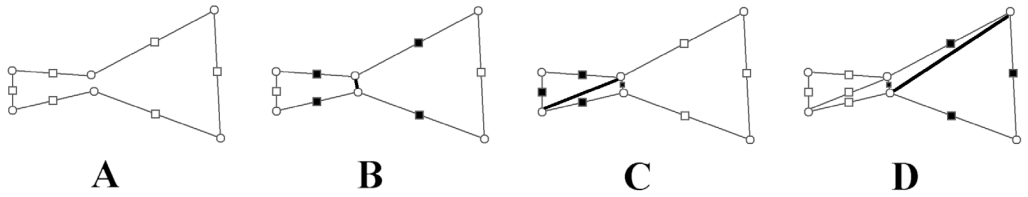

Figure 5.5: Constructing four triangular patches from six face-intersection curves. Original polygon is shown in image A. Circles are endpoints and squares are center control points of face-intersection curves. Dark lines indicate chosen diagonal for each split. Dark squares are control points used to determine center control point for each diagonal. Image B shows first diagonal selection. Image C shows diagonal selection for left half. Image D shows diagonal selection for right half.

each diagonal, only the center control points that are immediate neighbors to the endpoints of the diagonal are considered. This approach always provides four control points, see Figure 5.5.

All unique averages of each pair, group of three, and all four control points, in addition to each of the control points themselves are considered. This combination produces fifteen unique possibilities. For each point $\mathbf{b}_1^i$ in this set, try to form a rational-quadratic curve $\mathbf{Q}_i(u)$ to represent the diagonal with endpoints $\mathbf{b}_0$ and $\mathbf{b}_2$. The weight w_1^i for the curve is computed by intersecting the line connecting $\mathbf{b}_1^i$ and $\mathbf{m}$ with the isosurface, where $\mathbf{m} = \frac{\mathbf{b}_0+\mathbf{b}_2}{2}$, see [50] for how to compute w_1^i. A point $\mathbf{b}_1^i$ is ignored if there is not exactly one intersection with the contour surface. Choose the control point $\mathbf{b}_1^i$ that produces the curve $\mathbf{Q}_i(u)$ having least error. The error for $\mathbf{Q}_i(u)$ is estimated by evaluating it at parameter values $u = \left\{\frac{1}{6}, \frac{2}{6}, \frac{4}{6}, \frac{5}{6}\right\}$ and then sampling the quadratic tetrahedron at these locations. An error estimate is obtained by summing the absolute difference between the sampled values and the contour value. If none of the control points can form a valid curve, then the diagonal is invalid and the tetrahedron is marked as containing a surface that is "too complex."

When a contour surface is too complex, the tetrahedron is subdivided to produce simpler surface components. These are the criteria that indicate when a surface is too complex:

1. There are no face-intersection curves, but there exists a pill-shaped surface completely enclosed inside the tetrahedron. (Worsey-Farin [50] showed how to determine whether or not there exists such a surface.)

2. All curves in a face-intersection group lie on the same face.

3. A surface bounded by more than three face-intersection curves cannot be split into patches.

5.3 Curved-quadratic Elements

An isosurface in a curved-quadratic element T is found by first finding the isosurface in parameter space and then transforming the curve (or surface) to physical space based on the curved mapping defined by T. This section focuses on how to perform the transformation—in the 2D and 3D cases—to obtain quartic curves and surfaces that represent the isosurface through a curved-quadratic element.

5.3.1 The 2D Case

An isosurface—in parameter space—is represented by a set of rational-quadratic curves. Each curve is considered independently, and is put through a transformation from parameter space to physical space. In Bernstein-Bézier form, the bivariate quadratic mapping $\mathbf{T}(u,v) : \mathbb{U}^2 \longrightarrow \mathbb{R}^2$ of the standard triangle in parameter space $T^{\mathbb{U}}$—having corners $(0,0)^{\mathrm{T}}$, $(1,0)^{\mathrm{T}}$, and $(0,1)^{\mathrm{T}}$—to a curved triangle in physical space having six control points $\mathbf{b}_{ij} \in \mathbb{R}^2, i,j \geq 0$, $i+j \leq 2$, is defined as

$$\mathbf{T}(u,v) \;=\; \sum_{j=0}^{2} \sum_{i=0}^{2-j} \mathbf{b}_{ij} B_{ij}^2(u,v). \tag{5.7}$$

Substituting Equation (5.3) into Equation (5.7) transforms $\mathbf{Q}(u)$ from parameter space to physical space, given by the mapping $\mathbf{T}(\mathbf{Q}(u)) : \mathbb{U}^1 \longrightarrow \mathbb{U}^2 \longrightarrow \mathbb{R}^2$. Re-arranging the terms, one obtains

$$\mathbf{T}(\mathbf{Q}(u)) \;=\; \frac{\mathbf{c}_0 + \mathbf{c}_1 u + \mathbf{c}_2 u^2 + \mathbf{c}_3 u^3 + \mathbf{c}_4 u^4}{1 + g_1 u + g_2 u^2 + g_3 u^3 + g_4 u^4}. \tag{5.8}$$

The coefficients $\mathbf{c}_i$ and g_j are omitted, since they are quite "involved." (However, one can easily compute these coefficients from the $\mathbf{b}_{ij}$ in Equation (5.7) and the $\mathbf{p}_i$ and w_1 in Equation (5.3) using a symbolic mathematical software package.)

The univariate rational-quartic curve $\mathbf{C}(u) : \mathbb{U}^1 \longrightarrow \mathbb{R}^2$ used to represent the contour curve in physical space, in Bernstein-Bézier form, having five control points $\mathbf{d}_i \in \mathbb{R}^2$

and five weights $m_i, 0 \leq i \leq 4$, $m_i \geq 0$, is defined as

$$\mathbf{C}(u) \;=\; \frac{\sum_{i=0}^{4} m_i \mathbf{d}_i B_i^4(u)}{\sum_{i=0}^{4} m_i B_i^4(u)}. \tag{5.9}$$

In order to represent Equation (5.8) by Equation (5.9), a conversion from the *power-basis* form to the *Bernstein-basis* form is needed. The quartic power-basis form is described by

$$\alpha_0 + \alpha_1 u + \alpha_2 u^2 + \alpha_3 u^3 + \alpha_4 u^4, \tag{5.10}$$

which can be written as

$$\begin{bmatrix} 1 & u & u^2 & u^3 & u^4 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix}, \tag{5.11}$$

and is abbreviated by $\mathbf{u}^{\mathrm{T}}\mathbf{a}$. The quartic Bernstein-basis form, having weights ω_i, is described by

$$\omega_0\beta_0 B_0^4(u) + \omega_1\beta_1 B_1^4(u) + \omega_2\beta_2 B_2^4(u) + \omega_3\beta_3 B_3^4(u) + \omega_4\beta_4 B_4^4(u), \tag{5.12}$$

which can be written as

$$\begin{bmatrix} \omega_0 B_0^4(u) & \omega_1 B_1^4(u) & \omega_2 B_2^4(u) & \omega_3 B_3^4(u) & \omega_4 B_4^4(u) \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & u & u^2 & u^3 & u^4 \end{bmatrix} \mathbf{M} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}, \tag{5.13}$$

and is abbreviated as $\mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{b}$, where

$$\mathbf{M} = \begin{bmatrix} \omega_0 & 0 & 0 & 0 & 0 \\ -4\omega_0 & 4\omega_1 & 0 & 0 & 0 \\ 6\omega_0 & -12\omega_1 & 6\omega_2 & 0 & 0 \\ -4\omega_0 & 12\omega_1 & -12\omega_2 & 4\omega_3 & 0 \\ \omega_0 & -4\omega_1 & 6\omega_2 & -4\omega_3 & \omega_4 \end{bmatrix}. \tag{5.14}$$

The conversion from the power basis to the Bernstein basis is given by

$$\begin{aligned} \mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{b} &= \mathbf{u}^{\mathrm{T}}\mathbf{a}, \\ \mathbf{M}\mathbf{b} &= \mathbf{a}, \text{ and} \\ \mathbf{b} &= \mathbf{M}^{-1}\mathbf{a}, \end{aligned} \tag{5.15}$$

where $\mathbf{M}^{-1}$ is given by

$$\mathbf{M}^{-1} = \begin{bmatrix} \frac{1}{\omega_0} & 0 & 0 & 0 & 0 \\ \frac{1}{\omega_1} & \frac{1}{4\omega_1} & 0 & 0 & 0 \\ \frac{1}{\omega_2} & \frac{1}{2\omega_2} & \frac{1}{6\omega_2} & 0 & 0 \\ \frac{1}{\omega_3} & \frac{3}{4\omega_3} & \frac{1}{2\omega_3} & \frac{1}{4\omega_3} & 0 \\ \frac{1}{\omega_4} & \frac{1}{\omega_4} & \frac{1}{\omega_4} & \frac{1}{\omega_4} & \frac{1}{\omega_4} \end{bmatrix}. \tag{5.16}$$

The weights m_i for curve $\mathbf{C}$ are found by converting the denominator of Equation (5.8) to the denominator of Equation (5.9). Using Equation (5.15), when the weights ω_i are set to one, the conversion is given by

$$\begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} 1 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}. \tag{5.17}$$

The weights are given as

$$\begin{aligned} m_0 &= 1, \\ m_1 &= w_1, \end{aligned}$$

$$\begin{aligned} m_2 &= \frac{1}{3}(1 + 2{w_1}^2), \\ m_3 &= w_1, \text{ and} \\ m_4 &= 1. \end{aligned} \tag{5.18}$$

The control points $\mathbf{d}_i$ for curve $\mathbf{C}$ are found by converting the numerator of Equation (5.8) to the numerator of Equation (5.9). Using Equation (5.15), when weights $\omega_i = m_i$, the conversion is given by

$$\begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{d}_3 \\ \mathbf{d}_4 \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \\ \mathbf{c}_4 \end{bmatrix}. \tag{5.19}$$

The control points are given as

$$\begin{aligned} \mathbf{d}_0 &= \mathbf{c}_0, \\ \mathbf{d}_1 &= \frac{1}{4}\frac{4\mathbf{c}_0 + \mathbf{c}_1}{w_1}, \\ \mathbf{d}_2 &= \frac{1}{2}\frac{2\mathbf{c}_0 + \mathbf{c}_1 + \frac{1}{3}\mathbf{c}_2}{\frac{1}{3}(1 + 2{w_1}^2)}, \\ \mathbf{d}_3 &= \frac{1}{4}\frac{4\mathbf{c}_0 + 3\mathbf{c}_1 + 2\mathbf{c}_2 + \mathbf{c}_3}{w_1}, \text{ and} \\ \mathbf{d}_4 &= \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \mathbf{c}_4. \end{aligned} \tag{5.20}$$

(In practice, these values are easily computed using a symbolic math package.)

Examining the transformation of the control net of $\mathbf{Q}(u)$—defined by the three points $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$—reveals some similarities between the knots (control net) of $\mathbf{Q}(u)$ and those of $\mathbf{C}(u)$. The similarities are found by transforming two tangent lines $\mathbf{T}_L$ and $\mathbf{T}_R$ from parameter space to physical space, where $\mathbf{T}_L$ is the line segment connecting $\mathbf{p}_1$ and $\mathbf{p}_0$ and $\mathbf{T}_R$ is the line segment connecting $\mathbf{p}_1$ and $\mathbf{p}_2$. Two quadratic curves in physical space represent these tangent lines, which are found by fitting two quadratic curves—$\mathbf{l}(u)$ and $\mathbf{r}(u)$—to the transformed points $\left\{\mathbf{T}(\mathbf{p}_0), \mathbf{T}\left(\frac{\mathbf{p}_0+\mathbf{p}_1}{2}\right), \mathbf{T}(\mathbf{p}_1)\right\}$ and $\left\{\mathbf{T}(\mathbf{p}_2), \mathbf{T}\left(\frac{\mathbf{p}_2+\mathbf{p}_1}{2}\right), \mathbf{T}(\mathbf{p}_1)\right\}$, respectively. The curves $\mathbf{l}(u)$ and $\mathbf{r}(u)$ are found by first constraining their endpoints such that $\mathbf{l}_0 = \mathbf{T}(\mathbf{p}_0)$, $\mathbf{l}_2 = \mathbf{T}(\mathbf{p}_1)$, $\mathbf{r}_0 = \mathbf{T}(\mathbf{p}_2)$, and $\mathbf{r}_2 = \mathbf{T}(\mathbf{p}_1)$. Then, center controls points

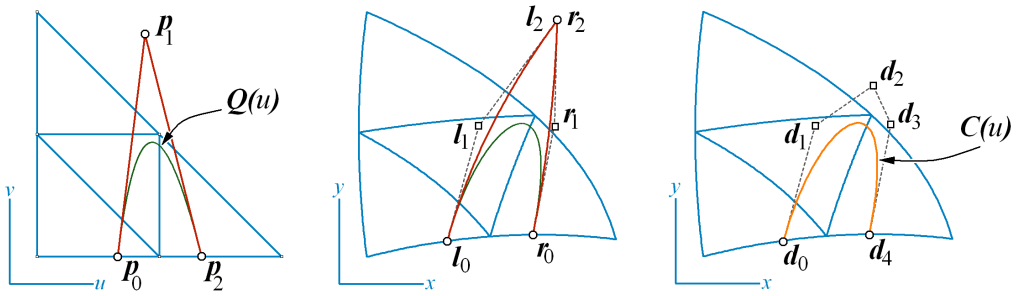

Figure 5.6: Relationship between control net of $\mathbf{Q}(u)$ and control net of $\mathbf{C}(u)$. Left image shows rational-quadratic curve $\mathbf{Q}(u)$ in parameter space. Middle image shows $\mathbf{Q}(u)$ transformed into physical space, together with the curves $\mathbf{l}(u)$ and $\mathbf{r}(u)$ and their control polygons. Right image shows rational-quartic curve $\mathbf{C}(u)$ resulting from transforming $\mathbf{Q}(u)$ into physical space. It turns out that $\mathbf{l}_1 = \mathbf{d}_1$, $\mathbf{r}_1 = \mathbf{d}_3$, $\mathbf{T}(\mathbf{p}_0) = \mathbf{d}_0$, and $\mathbf{T}(\mathbf{p}_2) = \mathbf{d}_4$.[3]

$\mathbf{l}_1$ and $\mathbf{r}_1$ are found by adding the constraints $\mathbf{l}\left(\frac{1}{2}\right) = \mathbf{T}\left(\frac{\mathbf{p}_0+\mathbf{p}_1}{2}\right)$ and $\mathbf{r}\left(\frac{1}{2}\right) = \mathbf{T}\left(\frac{\mathbf{p}_2+\mathbf{p}_1}{2}\right)$, which results in the solutions

$$\mathbf{l}_1 = 2\mathbf{T}\left(\frac{\mathbf{p}_0+\mathbf{p}_1}{2}\right) - \frac{\mathbf{T}(\mathbf{p}_0)+\mathbf{T}(\mathbf{p}_1)}{2} \text{ and} \tag{5.21}$$

$$\mathbf{r}_1 = 2\mathbf{T}\left(\frac{\mathbf{p}_2+\mathbf{p}_1}{2}\right) - \frac{\mathbf{T}(\mathbf{p}_2)+\mathbf{T}(\mathbf{p}_1)}{2}. \tag{5.22}$$

The center control points for $\mathbf{l}(u)$ and $\mathbf{r}(u)$ turn out to be $\mathbf{d}_1$ and $\mathbf{d}_3$, respectively, and $\mathbf{d}_0$ and $\mathbf{d}_4$ turn out to be $\mathbf{p}_0$ and $\mathbf{p}_2$ transformed to physical space, respectively, see Figure 5.6. This property is proved in the Appendix, see Section C.

Using this observation, four of the five required points are obtained that define the control net of $\mathbf{C}(\mathbf{u})$, given by

$$\begin{aligned} \mathbf{d}_0 &= \mathbf{T}(\mathbf{p}_0), \\ \mathbf{d}_1 &= \mathbf{l}_1, \\ \mathbf{d}_3 &= \mathbf{r}_1, \text{ and} \\ \mathbf{d}_4 &= \mathbf{T}(\mathbf{p}_2), \end{aligned} \tag{5.23}$$

while $\mathbf{d}_2$ is found using Equation (5.20).

[3]In this dissertation, quadratic elements are often rendered with parametric lines in the interior of the element. These lines help show the interior curvature of the element.

5.3.2 The 3D Case

An isosurface—in parameter space—is represented by a set of rational-quadratic patches. Each patch is considered independently, and is put through a curved quadratic transformation from parameter space to physical space. In Bernstein-Bézier form, the trivariate quadratic mapping $\mathbf{T}(u,v,w) : \mathbb{U}^3 \longrightarrow \mathbb{R}^3$ of the standard tetrahedron in parameter space—having corners $(0,0,0)^{\mathrm{T}}$, $(1,0,0)^{\mathrm{T}}$, $(0,1,0)^{\mathrm{T}}$, and $(0,0,1)^{\mathrm{T}}$—to a curved tetrahedron having ten control points $\mathbf{b}_{ijk} \in \mathbb{R}^3, i,j,k \geq 0,\ i+j+k \leq 2$, is defined as

$$\mathbf{T}(u,v,w) \ = \ \sum_{k=0}^{2} \sum_{j=0}^{2-k} \sum_{i=0}^{2-k-j} \mathbf{b}_{ijk} B_{ijk}^{2}(u,v,w), \tag{5.24}$$

where the trivariate n^{th}-degree Bernstein polynomial $B_{ijk}^{n}(u,v,w)$ is

$$B_{ijk}^{n}(u,v,w) \ = \ \frac{n!}{(n-i-j-k)!i!j!k!}(1-u-v-w)^{n-i-j-k}u^i v^j w^k. \tag{5.25}$$

Substituting Equation (5.6) into (5.24) transforms $\mathbf{Q}(u,v)$ from parameter space to physical space, $\mathbf{T}(\mathbf{Q}(u,v)) : \mathbb{U}^2 \longrightarrow \mathbb{U}^3 \longrightarrow \mathbb{R}^3$. This mapping is defined as

$$\mathbf{T}(\mathbf{Q}(u,v)) = \frac{\begin{array}{l}\mathbf{c}_0 + \mathbf{c}_1 u + \mathbf{c}_2 uv + \mathbf{c}_3 uv^2 + \mathbf{c}_4 uv^3 + \mathbf{c}_5 u^2 + \mathbf{c}_6 u^2 v + \mathbf{c}_7 u^2 v^2 + \\ \mathbf{c}_8 u^3 + \mathbf{c}_9 u^3 v + \mathbf{c}_{10} u^4 + \mathbf{c}_{11} v + \mathbf{c}_{12} v^2 + \mathbf{c}_{13} v^3 + \mathbf{c}_{14} v^4\end{array}}{\begin{array}{l}g_0 + g_1 u + g_2 uv + g_3 uv^2 + g_4 uv^3 + g_5 u^2 + g_6 u^2 v + g_7 u^2 v^2 + \\ g_8 u^3 + g_9 u^3 v + g_{10} u^4 + g_{11} v + g_{12} v^2 + g_{13} v^3 + g_{14} v^4\end{array}} \tag{5.26}$$

The coefficients $\mathbf{c}_i$ and g_j are omitted here, since they are quite complicated. (However, one can easily compute these coefficients from the $\mathbf{b}_{ijk}$ in Equation (5.24) and the $\mathbf{p}_{ij}$ and w_{10}, w_{01}, and w_{11} in Equation (5.6) using a symbolic mathematical software package.)

The bivariate rational-quartic surface $\mathbf{C}(u,v) : \mathbb{U}^2 \longrightarrow \mathbb{R}^3$ used to represent the contour surface in physical space, in Bernstein-Bézier form, having fifteen control points $\mathbf{d}_{ij} \in \mathbb{R}^3$ and fifteen weights $m_{ij}, i,j \geq 0,\ i+j \leq 4,\ m_{ij} \geq 0$, is defined as

$$\mathbf{C}(u,v) \ = \ \frac{\sum_{j=0}^{4} \sum_{i=0}^{4-j} m_{ij} \mathbf{d}_{ij} B_{ij}^{4}(u,v)}{\sum_{j=0}^{4} \sum_{i=0}^{4-j} m_{ij} B_{ij}^{4}(u,v)}. \tag{5.27}$$

The generalization of Equation (5.15) to the bivariate case is used to convert Equation (5.26) to Equation (5.27). Thus, the parametrization of $\mathbf{C}(u,v)$ is given by the

values

$$\begin{aligned}
m_{00} &= 1, \\
m_{10} &= w_{10}, \\
m_{20} &= \frac{1}{3}(1 + 2{w_{10}}^2), \\
m_{30} &= w_{10}, \\
m_{40} &= 1, \\
m_{01} &= w_{01}, \\
m_{11} &= \frac{1}{3}(w_{11} + 2w_{10}w_{01}), \\
m_{21} &= \frac{1}{3}(w_{01} + 2w_{10}w_{11}), \\
m_{31} &= w_{11}, \\
m_{02} &= \frac{1}{3}(1 + 2{w_{01}}^2), \\
m_{12} &= \frac{1}{3}(w_{10} + 2w_{01}w_{11}), \\
m_{22} &= \frac{1}{3}(1 + 2{w_{11}}^2), \\
m_{03} &= w_{01}, \\
m_{13} &= w_{11}, \text{ and} \\
m_{04} &= 1, \qquad (5.28)
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{d}_{00} &= \mathbf{c}_0, \\
\mathbf{d}_{10} &= \frac{1}{4}\frac{4\mathbf{c}_0 + \mathbf{c}_1}{w_{10}}, \\
\mathbf{d}_{20} &= \frac{1}{2}\frac{2\mathbf{c}_0 + \mathbf{c}_1 + \frac{1}{3}\mathbf{c}_5}{\frac{1}{3}(1 + 2{w_{10}}^2)}, \\
\mathbf{d}_{30} &= \frac{1}{4}\frac{4\mathbf{c}_0 + 3\mathbf{c}_1 + 2\mathbf{c}_5 + \mathbf{c}_8}{w_{10}}, \\
\mathbf{d}_{40} &= \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_5 + \mathbf{c}_8 + \mathbf{c}_{10}, \\
\mathbf{d}_{01} &= \frac{1}{4}\frac{4\mathbf{c}_0 + \mathbf{c}_{11}}{w_{01}}, \\
\mathbf{d}_{11} &= \frac{1}{4}\frac{4\mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_{11} + \frac{1}{3}\mathbf{c}_2}{\frac{1}{3}(2w_{10}w_{01} + w_{11})},
\end{aligned}$$

$$
\begin{aligned}
\mathbf{d}_{21} &= \frac{1}{4}\frac{4\mathbf{c}_0 + 2\mathbf{c}_1 + \mathbf{c}_{11} + \frac{2}{3}(\mathbf{c}_2 + \mathbf{c}_5) + \frac{1}{3}\mathbf{c}_6}{\frac{1}{3}(2w_{10}w_{11} + w_{01})}, \\
\mathbf{d}_{31} &= \frac{1}{4}\frac{4\mathbf{c}_0 + 3\mathbf{c}_1 + 2\mathbf{c}_5 + \mathbf{c}_6 + \mathbf{c}_2 + \mathbf{c}_{11} + \mathbf{c}_8 + \mathbf{c}_9}{w_{11}}, \\
\mathbf{d}_{02} &= \frac{1}{2}\frac{2\mathbf{c}_0 + \mathbf{c}_{11} + \frac{1}{3}\mathbf{c}_{12}}{\frac{1}{3}(1 + 2{w_{01}}^2)}, \\
\mathbf{d}_{12} &= \frac{1}{4}\frac{4\mathbf{c}_0 + 2\mathbf{c}_{11} + \mathbf{c}_1 + \frac{2}{3}(\mathbf{c}_2 + \mathbf{c}_{12}) + \frac{1}{3}\mathbf{c}_3}{\frac{1}{3}(2w_{01}w_{11} + w_{10})}, \\
\mathbf{d}_{22} &= \frac{1}{2}\frac{2\mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_{11} + \frac{2}{3}\mathbf{c}_2 + \frac{1}{3}(\mathbf{c}_3 + \mathbf{c}_5 + \mathbf{c}_6 + \mathbf{c}_7 + \mathbf{c}_{12})}{\frac{1}{3}(1 + 2{w_{11}}^2)}, \\
\mathbf{d}_{03} &= \frac{1}{4}\frac{3\mathbf{c}_{11} + 4\mathbf{c}_0 + \mathbf{c}_{13} + 2\mathbf{c}_{12}}{w_{01}}, \\
\mathbf{d}_{13} &= \frac{1}{4}\frac{\mathbf{c}_4 + \mathbf{c}_{13} + \mathbf{c}_2 + \mathbf{c}_3 + 2\mathbf{c}_{12} + \mathbf{c}_1 + 3\mathbf{c}_{11} + 4\mathbf{c}_0}{w_{11}}, \text{ and} \\
\mathbf{d}_{04} &= \mathbf{c}_0 + \mathbf{c}_{11} + \mathbf{c}_{12} + \mathbf{c}_{14} + \mathbf{c}_{13}.
\end{aligned} \tag{5.29}
$$

(In practice, these values are easily computed using a symbolic math package.)

5.4 Examples

An isosurface of a curved-quadratic tetrahedral representation of the "spherical data set" ($x^2+y^2+z^2 = c$) is shown in Figures 5.7, 5.8, and 5.9. Data set was created by first approximating $F(x, y, z) = x^2 + y^2 + z^2$ using the linear-edge quadratic tetrahedra approximation method described in Chapter 3 and then twisting the mesh to form curved-quadratic tetrahedra. This data set consists of 320 curved-quadratic tetrahedra. The extracted isosurface consists of 308 triangular rational-quartic patches.

Figures 5.10 and 5.11 show the isosurface of a data set consisting of 15918 quadratic tetrahedra representing "eight spheres." The curved quadratic-tetrahedral mesh uses the same 90° twist as the one shown in Figure 5.7. The resulting contour surfaces consist of 6112 patches.

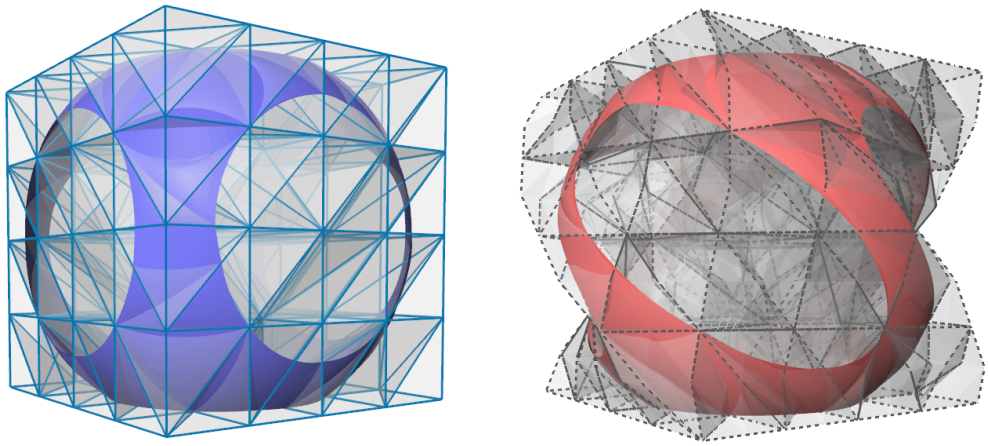

Figure 5.7: Left image shows "un-twisted" mesh containing only linear-edge quadratic tetrahedra. Right image shows twisted mesh containing curved-quadratic tetrahedra. The mesh is twisted by 90° when comparing orientations of top and bottom faces of configuration.

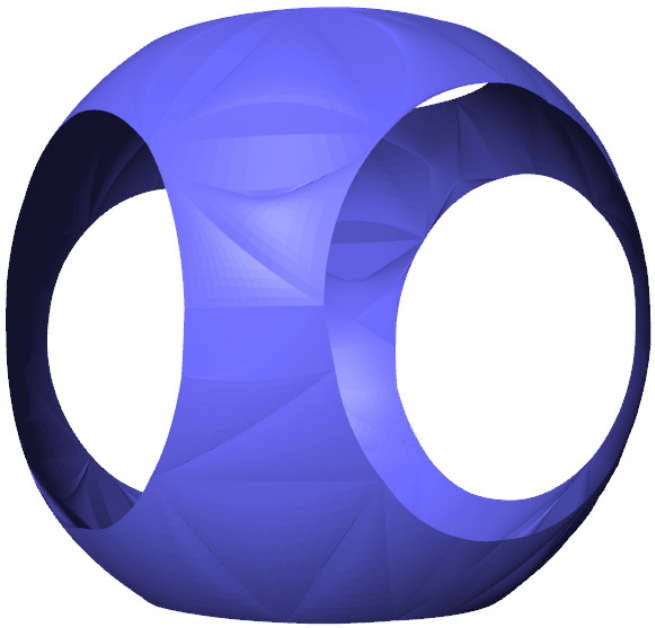

Figure 5.8: Magnification of piecewise rational-quadratic contour surface extracted from original mesh shown in Figure 5.7 (left); 320 quadratic tetrahedra.

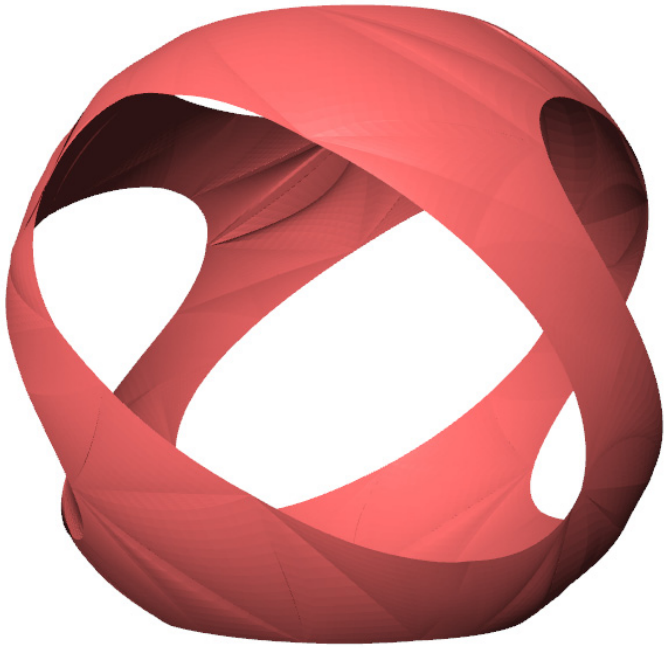

Figure 5.9: Magnification of piecewise rational-quartic contour surface extracted from twisted mesh shown in Figure 5.7 (right); 320 curved-quadratic tetrahedra.

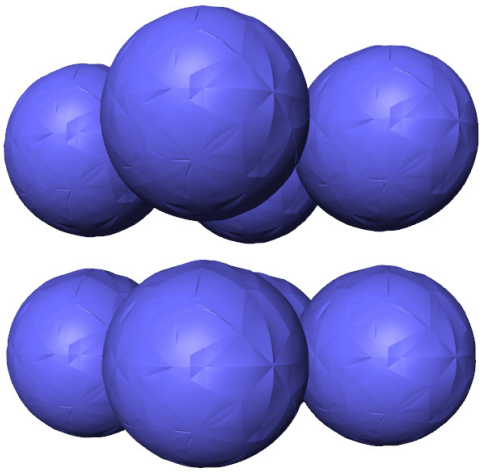

Figure 5.10: Piecewise rational-quadratic contour surface extracted from original mesh consisting of 15918 quadratic tetrahedra.

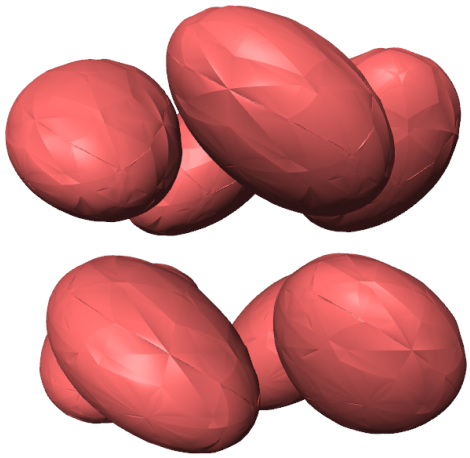

Figure 5.11: Piecewise rational-quartic contour surface extracted from 15918 curved-quadratic tetrahedra.

Chapter 6

Ray Casting

A fundamental rendering technique for tetrahedral elements (and all other types of volumetric mesh elements) is ray casting, see [30, 37]. The basic idea is to "shoot" rays—from a viewpoint to each pixel on an image plane—into a mesh of tetrahedra and color each pixel by accumulating *intersection segments* that result from intersecting the ray with elements in the mesh, see Figure 6.1. Many implementations of ray casting sample the data being visualized at discrete locations along the ray. This method works well for linear-edge elements, since it is relatively easy to determine where, inside an element, a sample point lies. Given a linear mapping T_{linear} from parameter space $\mathbb{U}$ to physical space $\mathbb{R}$ for an element, one puts the point $\mathbf{p} \in \mathbb{R}$ through the inverse transform T_{linear}^{-1} to find its parameter space tuple $\mathbf{u} \in \mathbb{U}$. The function overlying the element is then evaluated at $\mathbf{u}$ to provide the sample. When considering curved elements, however, determining a parameter space coordinate is non-trivial, since it is difficult to determine the inverse of a higher-order mapping.

While it is possible—and recommended—to discretely sample linear-edge quadratic elements along a ray (since a linear mapping defines the transformation from parameter space to physical space), a more cumbersome method of finding a close approximation to the actual intersection—between a ray and a quadratic element—is discussed as a foundation for intersecting a ray with a curved-quadratic element. Thus, the ray casting discussion for quadratic elements is limited to this problem: intersecting a line with a quadratic and curved-quadratic element. (This discussion assumes that the elements being visualized are

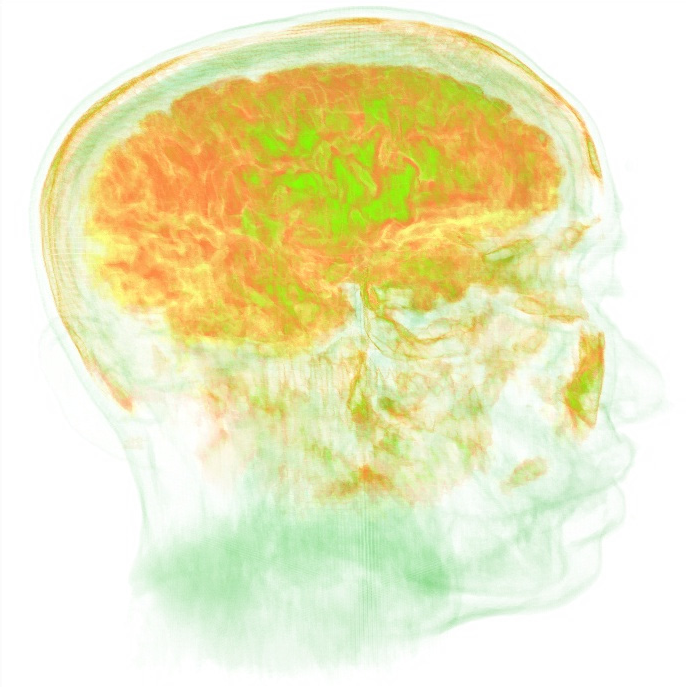

Figure 6.1: Ray casting of an MRI (magnetic resonance imaging) data set of human head.

valid and do not self-intersect.)

Ray casting is easily implemented by sampling the data set being visualized uniformly along a ray. For each sample point, one finds the element it lies in and then evaluates the polynomial overlying the containing element at that point to provide the sample. For a rectilinear grid, it is trivial to find the voxel (element) in which a sample point lies, and also to find the parameter-space (barycentric) coordinates of that point with respect to that element. These parameter space coordinates are needed to evaluate the polynomial over the containing element.

In the case of curved-quadratic tetrahedra, it is difficult to determine which element (in a mesh of curved-quadratic tetrahedra) contains a sample point. Even if it were known which curved-quadratic tetrahedron contained the point, it is difficult to obtain the parameter-space coordinates for that point with respect to the containing curved element, since its domain is defined by a curved mapping.

To provide samples along a ray, it is easier to intersect a line with the faces of a curved-quadratic tetrahedron to find the intersection segments that lie inside the tetrahedron. In physical space, these segments are straight lines, since they follow the ray.

However, looking at these segments in parameter space shows that they curve through the standard tetrahedron.

A method to construct a quadratic curve that approximates the intersection segment as it curves through parameter space is described in the following sections. Using this curve, one can sample along the curve to provide parameter-space coordinates that are used to sample the polynomial defined over the curved tetrahedron.

To reduce the time required to sample the polynomial defined over the curved element, an additional quadratic curve is found that approximates the polynomial overlying the intersection segment.

6.1 Linear-edge Quadratic Elements

Finding the intersection of a line with a quadratic element is more complicated than with a linear element. The goal is to find a representation $C^c(t) : \mathbb{U} \longrightarrow \mathbb{C}$ of the polynomial defined over the intersection segment. To better understand the 3D problem, the intersection between a line $l(s) : \mathbb{U} \longrightarrow \mathbb{R}^2$ and a linear-edge quadratic triangle $T(u, v) : \mathbb{U}^2 \longrightarrow \mathbb{R}^2$ is studied first.

6.1.1 The 2D Case

A quadratic curve $C^c(t)$ is used to approximate the polynomial defined over the intersection segment. The curve $C^c(t) : \mathbb{U} \longrightarrow \mathbb{C}$, having three coefficients $c_i,\ 0 \leq i \leq 2$, distributed uniformly across its domain $t \in [0, 1]$, is defined as

$$C^c(t) = \sum_{i=0}^{2} c_i B_i^2(t), \tag{6.1}$$

where B_i^2 is given by Equation (5.2).

There are two steps to finding an approximation to the intersection. First, intersect $l(s)$ with the boundary of $T(u, v)$. (This intersection is easily computed, since the boundaries are linearly defined in physical space.) There are three possibilities when intersecting a line with a triangle:

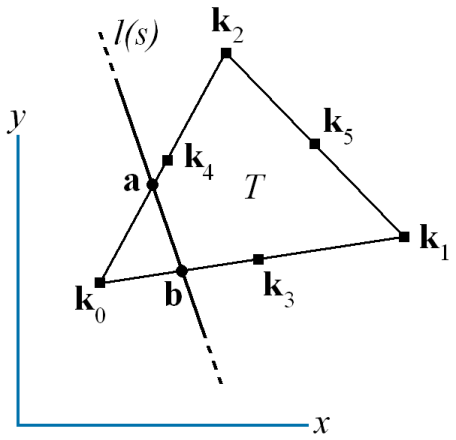

Figure 6.2: Intersection of line $l(s)$ with linear-edge quadratic triangle $T(u, v)$ at points **a** and **b**.

1. **No intersection.** The line does not intersect the triangle. Thus, this triangle does not contribute any information.

2. **One intersection.** The line intersects one of the corner knots, however, since such a small portion of the triangle intersects the ray, it does not contribute any information.

3. **Two intersections.** The line intersects two of the three boundary edges, forming one segment. This type of intersection is the only one that contributes information to the ray. (This includes the case when an edge is collinear with $l(s)$.)

Thus, there are at most two intersection points $\mathbf{a} = (x_a, y_a)^{\mathrm{T}} = T(\mathbf{u}_a)$ and $\mathbf{b} = (x_b, y_b)^{\mathrm{T}} = T(\mathbf{u}_b)$, see Figure 6.2, where $\mathbf{u}_a$ and $\mathbf{u}_b$ are the parameter space tuples of **a** and **b**, respectively, which are found by using the inverse of T, such that $\mathbf{u}_a = T^{-1}(\mathbf{a})$ and $\mathbf{u}_b = T^{-1}(\mathbf{b})$. Second, fit $C^c(t)$ to the polynomial defined over T (abbreviated as T^c) sampled at locations $\left\{\mathbf{u}_a, \frac{\mathbf{u}_a+\mathbf{u}_b}{2}, \mathbf{u}_b\right\}$. The approximation curve $C^c(t)$ is required to interpolate the endpoint values $T^c(\mathbf{u}_a)$ and $T^c(\mathbf{u}_b)$ so that neighboring-element intersections are at least C^0-continuous. Thus, $c_0 = T^c(\mathbf{u}_a)$, $c_2 = T^c(\mathbf{u}_b)$, and c_1—constrained by having $C\left(\frac{1}{2}\right) = T^c\left(\frac{\mathbf{u}_a+\mathbf{u}_b}{2}\right)$—is defined as

$$c_1 = 2\ T^c\left(\frac{\mathbf{u}_a + \mathbf{u}_b}{2}\right) - \frac{T^c(\mathbf{u}_a) + T^c(\mathbf{u}_b)}{2}. \tag{6.2}$$

(In the case where a quadratic approximation does not produce an accurate enough representation of the intersection, one can alternatively use a rational-quadratic or a higher-order

curve—such as cubic or quartic—to represent an intersection segment.)

6.1.2 The 3D Case

Ray casting a linear-edge quadratic tetrahedron is a straightforward extension of the 2D case. The only difference is the intersection points **a** and **b** are found by intersecting line $l(s)$ with the planar faces of a quadratic tetrahedron $T(u, v)$. As in the 2D case, there can be at most two intersection points, thus, only one intersection segment per ray. The approximation curve $C^c(t)$ is computed using the same method as in the 2D case.

A ray casting of a skull data set consisting of 5348 linear-edge quadratic tetrahedra is shown in Figure 6.3. Light regions show low-density areas and dark regions show high-density areas in the data set. (Similar to the inverse of an x-ray.)[1] The image-space error is 1.96%, see the Appendix, Section B. Back-to-front compositing was used to accumulate several uniform samples along each ray. A pixel intensity $I_n^{i,j}$ computed from the n^{th} sample $c_n \in [0, 1]$ is given by

$$I_n^{i,j} = I_{n-1}^{i,j} D(1 - c_n), \tag{6.3}$$

where D is the sampling distance and $(i, j)^{\mathrm{T}}$ is the location of the pixel. The samples in the quadratic case were taken from the intersection segments found along each ray. The samples in the "original" case were taken directly from the rectilinear data set.

6.2 Curved-quadratic Elements

Ray casting curved elements is more difficult than ray casting linear-edge elements, since 1) there can be more than one intersection segment per element and 2) it is difficult to represent the polynomial overlying the intersection segment. The method to intersect a line with a curved-quadratic element is first discussed in the 2D case and then extended to the 3D case.

[1] The 305×311 pixel image required one hour to compute on a 2.8GHz Pentium IV graphics workstation with 2GB of main memory. The "original" image—computed from a rectilinear scalar field consisting of 278528 data sites—required one minute to compute. (No optimization was performed for the quadratic tetrahedron implementation; the brute force method of testing all elements for intersection with each ray was used.)

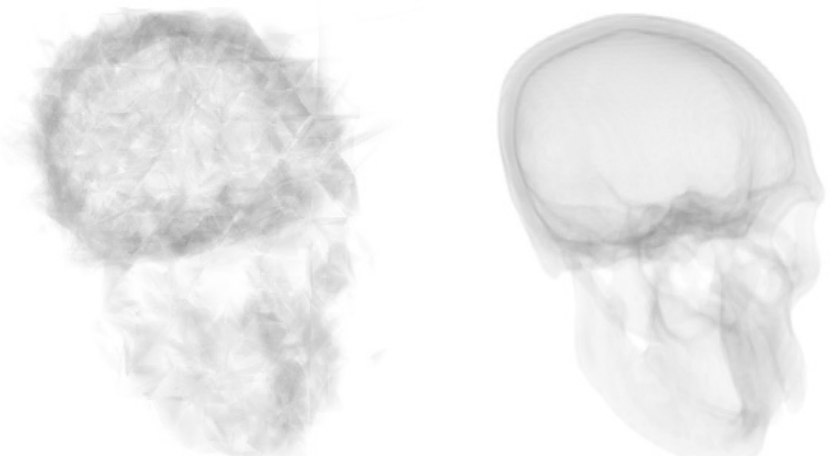

Figure 6.3: Ray casting of skull data set. Left image shows an approximation consisting of 5348 linear-edge quadratic tetrahedra and 7487 knots. Right image shows original data set consisting of 278528 data sites. Light and dark regions show low- and high-density areas in the data set, respectively. The image-space error for the approximation is 1.96%.

6.2.1 The 2D Case

A line $l(s) : \mathbb{U} \longrightarrow \mathbb{R}^2$ intersects the boundary of a curved-quadratic triangle $T(u, v) : \mathbb{U}^2 \longrightarrow \mathbb{R}^2$ in at most six locations (not considering degenerate cases). The goal is to find segments—between intersection points—that lie inside $T(u, v)$. To compute the intersections between $l(s)$ and the boundary edges of $T(u, v)$ one must intersect $l(s)$ with three quadratic curves—the boundary edges of $T(u, v)$. One might consider finding the inverse $l^{-1}(t) : \mathbb{U} \longrightarrow \mathbb{U}^2$ of $l(s)$ based on the mapping $T(u, v)$. In this case, $l^{-1}(t)$ could then be intersected with the standard triangle in parameter space, see Figure 6.4. It is possible to find a closed form representation of the inverse of $T(u, v)$ in 2D, however, since $T(u, v)$ is not always bijective, there may exist none, more than one solution, or imaginary solutions, for a given point. In 3D it is not possible to find a closed form representation. Thus, a method that is extensible to higher dimensions is desired. The method to find an approximation $C(t)$ to $l^{-1}(s)$ based on a curved-quadratic mapping $T(u, v)$ has four steps:

1. Intersect $l(s)$ with the three boundary curves of $T(u, v)$ to produce a set of N intersection points $\mathbf{p}_m = (x_m, y_m)^{\mathrm{T}},\ 0 \leq m \leq N - 1$.

2. Reorder the points $\mathbf{p}_m$ sequentially based upon the distance from the viewpoint.

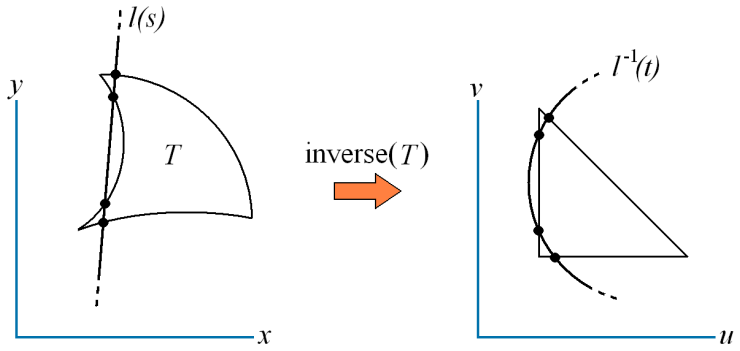

Figure 6.4: Transformation of line $l(s)$ in xy-space to $l^{-1}(t)$ in uv-space by the inverse transformation of the curved-quadratic triangle $T(u, v)$.

3. Form intersection segments from sequentially adjacent pairs of points.

4. Discard invalid segments by testing whether the segment is inside or outside of the triangle $T(u, v)$.

Each of these four steps is described in detail in the following paragraphs.

Step 1

The intersection of line $l(s)$ with quadratic curve $Q_e(t)$—representing edge e of curved-quadratic triangle $T(u, v)$—can be computed analytically or iteratively, see [4]. (An iterative method is more stable than an analytical method and the speed of the iterative method (i.e., number of iterations used) is inversely proportional to the accuracy of the result.) Using the analytical approach, assuming $Q_e(t)$ is in the form

$$Q(t) = \mathbf{r}_0 t^2 + \mathbf{r}_1 t + \mathbf{r}_2, \tag{6.4}$$

and $l(s)$ is in the form

$$l(s) = \mathbf{q}_0 + \mathbf{q}_1 s, \tag{6.5}$$

where $\mathbf{r}_i = (x_i^r, y_i^r)^{\mathrm{T}},\ 0 \le i \le 2$, and $\mathbf{q}_j = (x_j^q, y_j^q)^{\mathrm{T}},\ 0 \le j \le 1$, the intersections of $Q(t)$ and $l(s)$ are given by the roots of

$$(x_0^r y_1^q - y_0^r x_1^q)t^2 + (x_1^r y_1^q - y_1^r x_1^q)t + x_2^r y_1^q + x_1^q y_0^q - x_0^q y_1^q - x_1^q y_2^r = 0 \tag{6.6}$$

and

$$s(t) = \frac{(r_x^0 r_y^1 - r_y^0 r_x^1)t + r_y^0 q_x^0 + r_y^2 r_x^0 - r_y^0 r_x^2 - q_y^0 r_x^0}{r_x^0 q_y^1 - r_y^0 q_x^1}. \tag{6.7}$$

Thus, there are two solution pairs $\{(t_i, s_i)\},\ 0 \leq i \leq 1$, where $s_i = s(t_i)$. Points not satisfying the constraint $0 \leq t_i \leq 1$ lie outside of the curved triangle and are discarded.

Step 2

Labelling the intersection points $\mathbf{p}_m$ based upon their distance from the viewpoint orders the points. This ordering is important, since only points that are adjacent sequentially can form an intersection segment. Since the parameter values s_i that result during Step 1 are directly related to the distance from the viewpoint, this ordering is easily determined.

Step 3

An approximation to $l^{-1}(t)$ is constructed by transforming two vectors—aligned with $l(s)$—to parameter space, so that they become tangent vectors to the quadratic curve $C(t)$ used to approximate $l^{-1}(t)$. The curve $C(t) : \mathbb{U} \longrightarrow \mathbb{U}^2$, having three knots $\mathbf{u}_i$ and three coefficients $c_i,\ 0 \leq i \leq 2$, distributed uniformly across its domain $t \in [0, 1]$, is defined as

$$C(t) = \sum_{i=0}^{2} \mathbf{u}_i B_i^2(t), \tag{6.8}$$

where B_i^2 is given by Equation (5.2).

A vector $\mathbf{v}$ in physical space—located at point $\mathbf{p} = T(\mathbf{u})$, where $\mathbf{u} \in \mathbb{U}^2$—is transformed to a vector $\mathbf{w}$ in parameter space by considering the linear combination $\mathbf{v} = \alpha\mathbf{X} + \beta\mathbf{Y}$, where $\mathbf{X} = T^u(\mathbf{u})$, $\mathbf{Y} = T^v(\mathbf{u})$, and $T^i(u, v)$ denotes the partial derivative of $T(u, v)$ with respect to the i^{th} direction. Solving for α and β yields the transformed vector $\mathbf{w} = (\alpha, \beta)^{\mathrm{T}}$, see Figure 6.5. (The inverse transformation of vectors is possible because the partial derivatives of $T(u, v)$ are linear functions.)

An intersection segment m is bounded by two sequentially adjacent points $\mathbf{p}_m$ and $\mathbf{p}_{m+1}$. An approximation to the inverse of the intersection segment is constructed by first transforming vectors $\mathbf{v}_m$ and $\mathbf{v}_{m+1}$ in physical space—which are aligned with $l(s)$ and positioned at $\mathbf{p}_m$ and $\mathbf{p}_{m+1}$, respectively—to vectors $\mathbf{w}_m$ and $\mathbf{w}_{m+1}$ in parameter space. (This step assumes parameter space coordinates $\mathbf{u}_m$ are known for points $\mathbf{p}_m$. These coordinates are computed in Step 1 from the solutions t_i for each edge.) The lines implied by vectors $\mathbf{w}_m$ and $\mathbf{w}_{m+1}$—while located at $\mathbf{u}_m$ and $\mathbf{u}_{m+1}$, respectively—are intersected

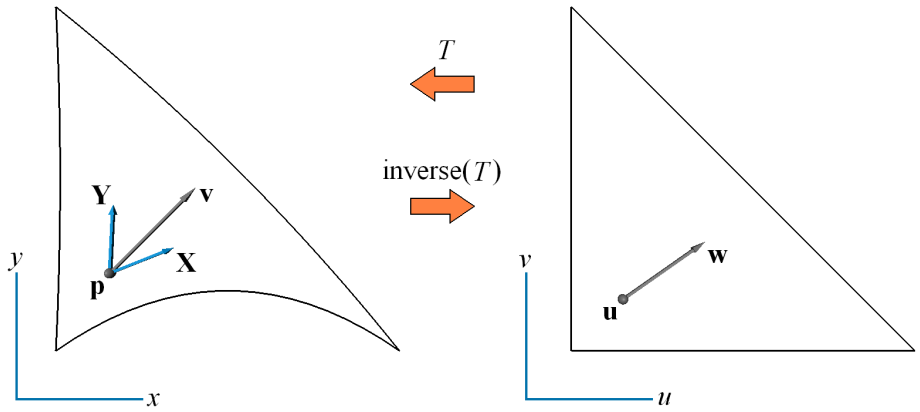

Figure 6.5: Inverse transformation of vector $\mathbf{v}$—located at point $\mathbf{p} = T(\mathbf{u})$—through curved-quadratic mapping $T(u, v)$. Vector $\mathbf{v}$ is transformed from physical space to vector $\mathbf{w}$ in parameter space by considering the linear combination $\mathbf{v} = \alpha\mathbf{X} + \beta\mathbf{Y}$, where $\mathbf{w} = (\alpha, \beta)^{\mathrm{T}}$.

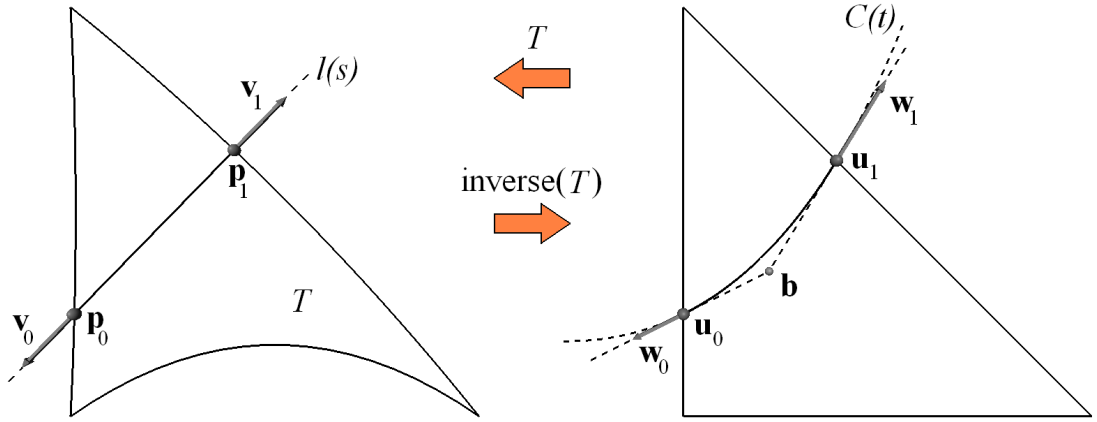

Figure 6.6: Quadratic curve $C(t)$ approximates the inverse of line $l(s)$ based on the curved-quadratic mapping $T(u, v)$.

to form a third point $\mathbf{b}_m$. The three points $\{\mathbf{u}_m, \mathbf{b}_m, \mathbf{u}_{m+1}\}$ are used as control points for $C_m(t) : \mathbb{U} \longrightarrow \mathbb{U}^2$, which represents an approximation to $l^{-1}(s)$, see Figure 6.6.

Next, the coefficients of $C_m(t)$ are computed by fitting the uniformly spaced coefficients c_i to the three samples $\left\{T^c\big(C(0)\big)\,,\ T^c\Big(C\left(\frac{1}{2}\right)\Big)\,,\ T^c\big(C(1)\big)\right\}$. They are given by

$$\begin{aligned} c_0 &= T^c\big(C(0)\big) = T^c(\mathbf{u}_i), \\ c_1 &= 2\,T^c\left(C\left(\frac{1}{2}\right)\right) - \frac{T^c(\mathbf{u}_i) + T^c(\mathbf{u}_{i+1})}{2}, \text{ and} \\ c_2 &= T^c\big(C(1)\big) = T^c(\mathbf{u}_{i+1}), \end{aligned} \tag{6.9}$$

where $T^c(u, v)$ evaluates the polynomial overlying $T(u, v)$.

Step 4

Each intersection segment C_m is checked to see whether it lies inside or outside of curved-quadratic triangle T. This is done by checking where the point $C_m(\frac{1}{2})$ lies relative to the standard triangle; if it lies outside, the intersection segment is also outside and does not contribute information to the ray.

6.2.2 The 3D Case

The ideas discussed in the 2D case are extended to the 3D case. The intersection between a ray and a curved-quadratic tetrahedron $T(u, v, w)$ reduces to the intersection between a ray and the curved surfaces (faces) of $T(u, v, w)$. Thus, rather than intersecting $l(s)$ with planar faces (as in the case of a linear-edge quadratic tetrahedron), $l(s)$ is intersected with four curved-triangular patches $Q_e(u, v) : \mathbb{U}^2 \longrightarrow \mathbb{R}^3,\ 0 \leq e \leq 3$. The four steps to compute an approximation $C_m(t) : \mathbb{U} \longrightarrow \mathbb{U}^3$ to an intersection segment m are the same as in the 2D case. Only the details of the differences for each step are discussed.

Step 1

There are several options for performing the intersection between $l(s)$ and the curved patch $Q_e(u, v)$. Tessellating the patch with several linear triangles, then, intersecting $l(s)$ with these triangles is used here. It is possible, with this method, to directly compute the parameter space coordinates $\mathbf{u}_m$ of the intersection points $\mathbf{p}_m$ simultaneously when computing the intersection between $l(s)$ and $Q_e(u, v)$.

Step 2

The intersection points $\mathbf{p}_m$ are sorted based upon the distance from the viewpoint. The parameter value s along $l(s)$ is used, since it was computed during the intersection process of Step 1.

Step 3

A vector $\mathbf{v}$ in physical space—located at point $\mathbf{p} = T(\mathbf{u})$, where $\mathbf{u} \in \mathbb{U}^3$—is transformed to a vector $\mathbf{w}$ in parameter space by considering the linear combination $\mathbf{v} = \alpha\mathbf{X} + \beta\mathbf{Y} + \gamma\mathbf{Z}$, where $\mathbf{X} = T^u(\mathbf{u})$, $\mathbf{Y} = T^v(\mathbf{u})$, $\mathbf{Z} = T^w(\mathbf{u})$, and $T^i(u, v, w)$ denotes the partial derivative of $T(u, v, w)$ with respect to the i^{th} direction. Solving for α, β, and γ gives the transformed vector $\mathbf{w} = (\alpha, \beta, \gamma)^{\mathrm{T}}$.

An intersection segment $C_m(t)$ is constructed using the same method as described in Step 3 of the 2D case, see Section 6.2.1. The only difference is that it is the exception for two lines in 3D to intersect. Rather than intersecting the two lines $\mathbf{l}_m$ and $\mathbf{l}_{m+1}$—implied by vectors $\mathbf{w}_m$ and $\mathbf{w}_{m+1}$, while positioned at $\mathbf{u}_m$ and $\mathbf{u}_{m+1}$, respectively—two points $\mathbf{g}_m$ and $\mathbf{g}_{m+1}$ are found so that $\mathbf{g}_m$ is the closest point on $\mathbf{l}_m$ to $\mathbf{l}_{m+1}$ and $\mathbf{g}_{m+1}$ is the closest point on $\mathbf{l}_{m+1}$ to $\mathbf{l}_m$. (Points $\mathbf{g}_m$ and $\mathbf{g}_{m+1}$ are computed by minimizing the distance between $\mathbf{l}_m$ and $\mathbf{l}_{m+1}$.) Point $\mathbf{b}_m$ is given by $\mathbf{b}_m = \frac{\mathbf{g}_m + \mathbf{g}_{m+1}}{2}$.

Step 4

The same method used in Step 4 of the 2D case is used to discard intersection segments that lie outside of the curved-quadratic tetrahedron, see Section 6.2.1.

6.3 Examples

A ray casting of a single curved-quadratic tetrahedron having uniform density—all coefficients have value one—is shown in Figure 6.7. The 467×569 pixel image required 12.5 minutes to compute. Ray intersections were performed by tessellating each "face" of the curved-quadratic tetrahedron with 400 linear triangles. (No optimization was performed; the brute force method of testing all triangles—from all four faces—for intersection with each ray was used.) The same accumulation method used in the linear-edge case was used here, see Section 6.1.2.

A ray casting of a single curved-quadratic tetrahedron having non-uniform density—where the corner coefficients are zero and edge coefficients are one—is shown in Figure 6.8. The 487×473 pixel image required 11.1 minutes to compute. Ray intersections were per-

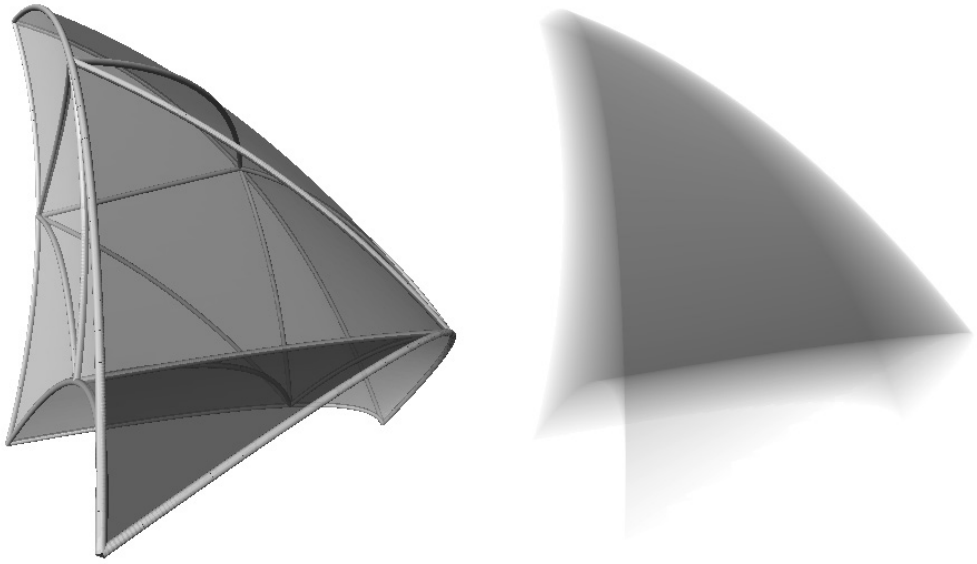

Figure 6.7: Ray casting of curved-quadratic tetrahedron $T(u, v, w)$ having coefficients $c_i = 1,\ 0 \leq i \leq 9$. Left image shows a rendering of the curved "faces" of $T(u, v, w)$. Right image shows ray casting of $T(u, v, w)$ (dark areas denoting higher density).[3]

formed by tessellating each "face" of the curved-quadratic tetrahedron with 400 linear triangles. The same accumulation method used in the linear-edge case was used here, and the density values were mapped so that three isosurfaces were visualized. Isosurfaces are for a small range of values near $\{0.2, 0.575, 0.725\}$. The "egg" corresponds to the isosurface at 0.725 and the corners correspond to 0.2.[2]

For both examples, the accumulation method used along each ray was the same method used in the linear-edge quadratic tetrahedron case, as described by Equation (6.3).

[2]Both examples were computed on a 2.8GHz Pentium IV graphics workstation with 2GB of main memory.

[3]In this dissertation, quadratic tetrahedra are often rendered with parametric lines on the faces of the element. These lines help show the curvature of the element.

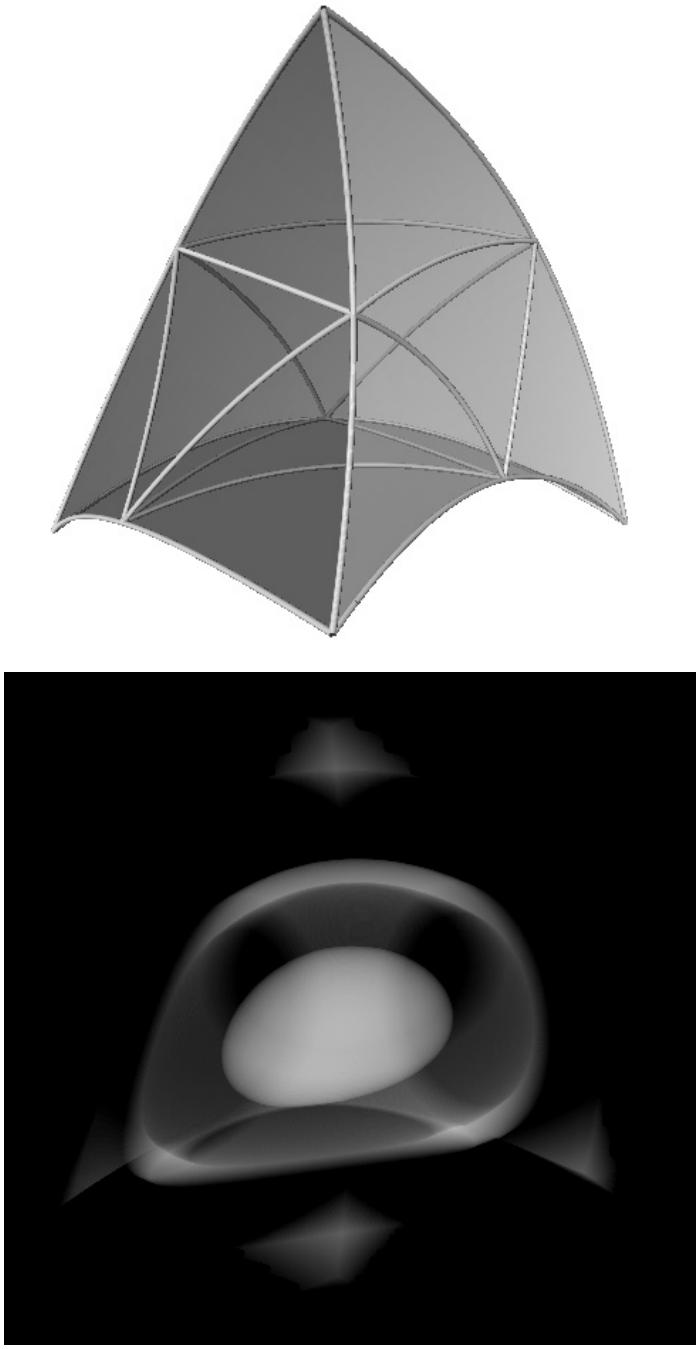

Figure 6.8: Ray casting of curved-quadratic tetrahedron $T(u, v, w)$ having non-uniform density—where corner coefficients are zero and edge coefficients are one. Top image shows a rendering of the curved "faces" of $T(u, v, w)$. Bottom image shows ray casting of $T(u, v, w)$. Isosurfaces are for a small range of values near $\{0.2, 0.575, 0.725\}$. The "egg" corresponds to the isosurface at 0.725 and the corners correspond to 0.2.

Chapter 7

Cutting Planes

A *cutting plane* is used to "cut" through a data set—usually a planar cut in the context of a 3D domain. A cutting plane visualization shows the intersection between a plane and a data set. This is useful when volume visualization, such as ray casting, is impractical because too much data needs to be processed. For example, consider examining various layers of the Earth's atmosphere. If a user wanted to examine all of the layers at once, instead of using volume visualization, it would be more useful to visualize the intersection of the data with, say, a plane passing through the equator.

There are many methods for the visualization of cutting planes. As in ray casting, one could simply sample the data at discrete locations across the plane. However, as in ray casting, the sampling method does not work well for curved elements, since it is difficult to determine the parameter space coordinate of an arbitrary point with respect to a curved element. Visualization of a cutting planes is done by intersecting elements independently of each other. Thus, the essence of cutting planes is to intersect an element T with cutting plane R. The actual intersection is computed by intersecting each of the edges of T with R, forming *edge-intersection points*. Then, the edge-intersection points are connected together—over each face—to form face-intersection curves (as in isosurfacing). Then, face-intersection curves are grouped together to form polygons that bound the intersection surface. In the case of linear elements, there is only one intersection segment (2D case) or surface (3D case). The surface in the 3D case may be bounded by either a triangle

or a quadrilateral.

While it is possible—and recommended—to discretely sample linear-edge quadratic elements across the cutting plane, a more cumbersome method of finding a close approximation to the actual intersection is discussed. The cutting plane discussion for quadratic elements is limited to this problem: intersecting a plane with a quadratic and curved-quadratic element. (This discussion assumes that the elements being visualized are valid and do not self-intersect or overlap with other elements.)

7.1 Linear-edge Quadratic Elements

The goal is to represent the intersection of quadratic tetrahedron T with cutting plane R by a set of quadratic triangles $\mathcal{T}$. The set $\mathcal{T}$ contains either zero, one, or two triangles, since the intersection of a plane with a linear-edge tetrahedron is limited to 1) no intersection, 2) an intersection forming a triangle, or 3) an intersection forming a quadrilateral (which is represented by two triangles).

Assuming T is defined by ten knots $\mathbf{k}_i,\ 0 \leq i \leq 9$, only the corner knots $0 \leq i \leq 3$ are considered, since the edge knots are positioned at the midpoints of the edges, see Figure 7.1. Thus, the six edges $e_j,\ 0 \leq j \leq 5$ of T are bounded by knot pairs $(0,1)$, $(0,2)$, $(0,3)$, $(1,2)$, $(1,3)$, and $(2,3)$. Line segments—formed between the corner knots defining each edge—are intersected with the R to yield a set of N intersection points $\mathbf{p}_m,\ 0 \leq m \leq N-1$, in physical space (the parameter space coordinate $\mathbf{u}_m$ for each $\mathbf{p}_m$ is also computed, such that $\mathbf{p}_m = T(\mathbf{u}_m)$). There are five values that N can take on:

- **Zero**. Plane R does not intersect T.
- **One**. Plane R intersects one corner knot of T.
- **Two**. One of the edges of T lies on R (only the endpoints of the intersection are counted).
- **Three**. Plane R intersects three edges of T.
- **Four**. Plane R intersects four edges of T.

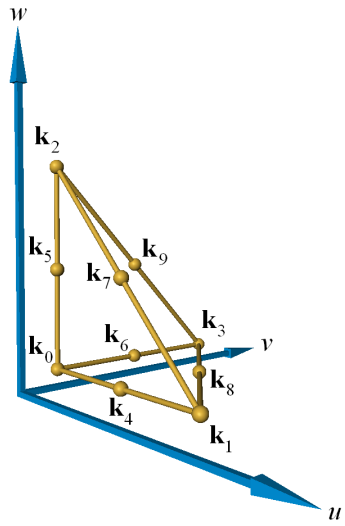

Figure 7.1: Indexing used for curved-quadratic tetrahedron.

(An intersection surface is produced only for the cases where N is three or four.) When N is three, a quadratic triangle $C(u, v)$—having six knots $\mathbf{v}_n$ and six coefficients c_n—is used to approximate the intersection and is formed by first setting its knots to the locations given by

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{u}_0, \\
\mathbf{v}_1 &= \mathbf{u}_1, \\
\mathbf{v}_2 &= \mathbf{u}_2, \\
\mathbf{v}_3 &= \frac{\mathbf{u}_0 + \mathbf{u}_1}{2}, \\
\mathbf{v}_4 &= \frac{\mathbf{u}_0 + \mathbf{u}_2}{2}, \text{ and} \\
\mathbf{v}_5 &= \frac{\mathbf{u}_1 + \mathbf{u}_2}{2}.
\end{aligned} \tag{7.1}$$

Then, the function overlying the intersection surface is approximated by defining the coefficients c_n of $C(u, v)$ as

$$\begin{aligned}
c_0 &= T^c(\mathbf{u}_0), \\
c_1 &= T^c(\mathbf{u}_1), \\
c_2 &= T^c(\mathbf{u}_2), \\
c_3 &= 2T^c\left(\frac{\mathbf{u}_0 + \mathbf{u}_1}{2}\right) - \frac{T^c(\mathbf{u}_0) + T^c(\mathbf{u}_1)}{2},
\end{aligned}$$

$$
\begin{aligned}
c_4 &= 2T^c\left(\frac{\mathbf{u}_0+\mathbf{u}_2}{2}\right) - \frac{T^c(\mathbf{u}_0)+T^c(\mathbf{u}_2)}{2}, \text{ and} \\
c_5 &= 2T^c\left(\frac{\mathbf{u}_1+\mathbf{u}_2}{2}\right) - \frac{T^c(\mathbf{u}_1)+T^c(\mathbf{u}_2)}{2},
\end{aligned} \tag{7.2}
$$

where $T^c(u,v,w)$ evaluates the polynomial defined over $T(u,v,w)$.

When N is four, an artificial diagonal is added to divide the quadrilateral into two triangles. This is done by choosing the shortest edge to use as the diagonal. (In the case where both diagonals are the same length, and one desires a unique solution, an additional point could be added in the middle of the quadrilateral to create four triangles.) The coefficients for the two triangles are found by first relabelling the intersection points $\mathbf{p}_m$ and $\mathbf{u}_m$ with respect to each triangle, then, using Equation (7.2) finds the coefficients.

7.2 Curved-quadratic Elements

It is more difficult to compute an approximation to the intersection of a cutting plane R with a curved-quadratic tetrahedron T than with a linear-edge quadratic tetrahedron. In this case, a close approximation to the exact intersection is left for future research because of the case shown in Figure 7.2 where R does not intersect any of the curved edges of tetrahedron T. However, a sampling method that leverages the ray casting method of the previous chapter is described. The plane R is discretized so that a set of parallel rays—lying on the plane—are passed through the data being visualized. A uniform rectilinear representation of the cutting plane is then constructed by discretely sampling each ray uniformly, see Figure 7.3. The cutting plane is visualized by rendering this uniform rectilinear representation.

7.3 Examples

Two cutting planes through a single curved-quadratic tetrahedron—where corner coefficients are zero and edge coefficients are one—are shown in Figure 7.4. Each cutting plane visualization used the ray casting method described in Section 6.2.2 as applied in Section 7.2 and was discretized to a 652×621 grid (producing an RGB pixel image of the same size). Each curved face of the tetrahedron was tessellated with 400 linear triangles for the intersec-

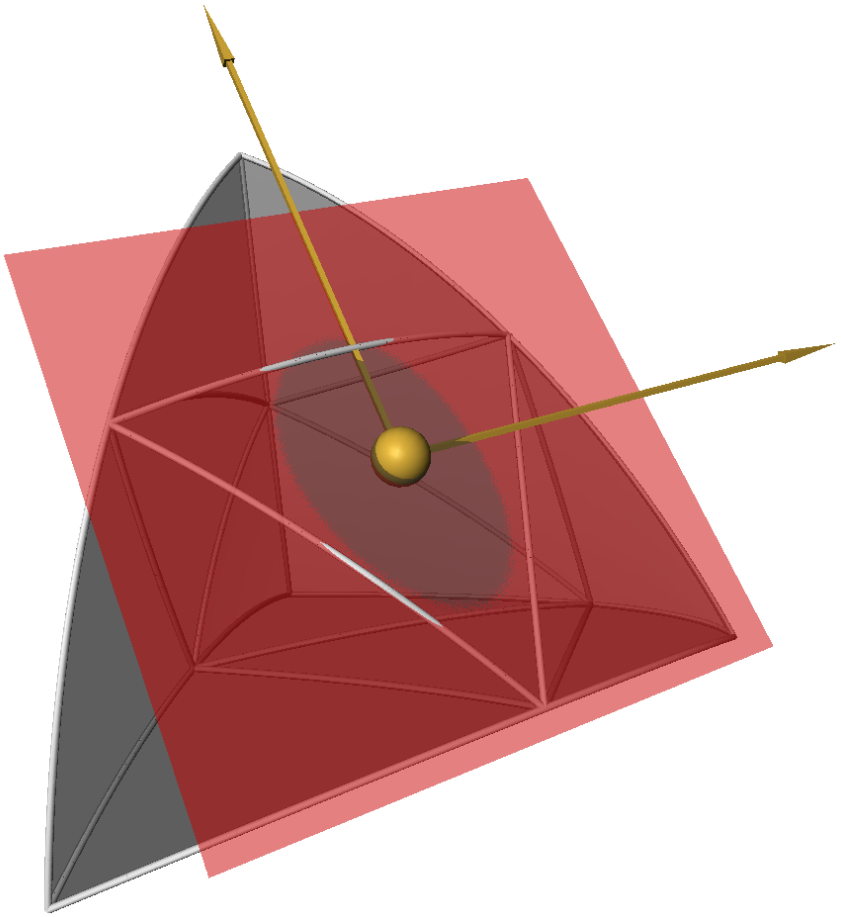

Figure 7.2: Intersection of plane and curved-quadratic tetrahedron. Difficulties arise since the plane can intersect the tetrahedron without intersecting its edges.[1]

tion testing.[2] The color map used starts from red, changing to orange, to yellow, to green, and finally to blue, spreading uniformly across the domain from zero to one, respectively. (Corners are red and the center is green-blue.)

[1] In this dissertation, quadratic tetrahedra are often rendered with parametric lines on the faces of the element. These lines help show the curvature of the element. Plane R intersects the parametric lines shown on the faces of tetrahedron T in Figure 7.2, however, these lines are not as useful as the edges of the tetrahedron.

[2] Each cutting plane visualization required 12 seconds to compute on a 2.8GHz Pentium IV graphics workstation with 2GB of main memory.

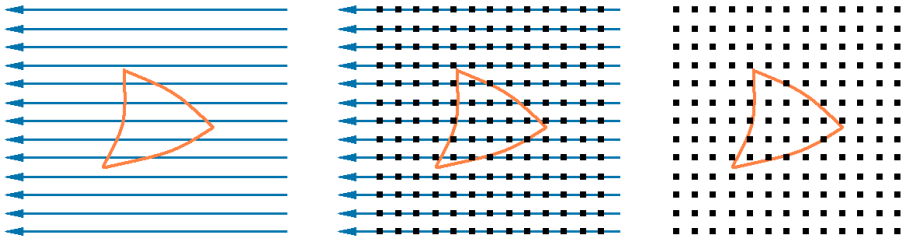

Figure 7.3: Discrete sampling of cutting plane. Rays are cast through data and then sampled discretely to construct a uniform rectilinear grid that represents the cutting plane. Left image shows rays—lying on the cutting plane—used to intersect elements. Middle image shows discretely sampled rays. Right image shows resulting uniform rectilinear grid used for visualization.

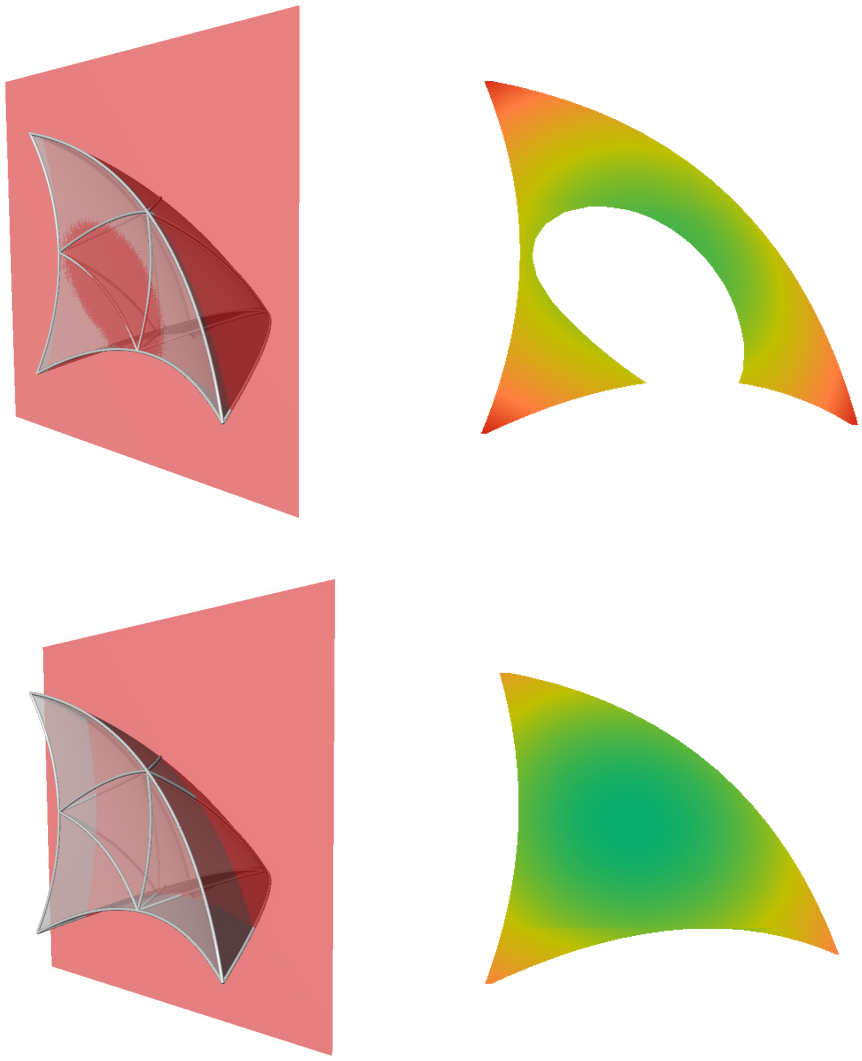

Figure 7.4: Two cutting planes intersecting curved-quadratic tetrahedron.

Chapter 8

Serendipity Element "Hex20"

Finite element methods commonly use higher-order so-called "serendipity" hexahedral elements [9]. The *Hex20 serendipity element* uses 20 Lagrange basis functions associated with its 20 knots and is parameterized by 20 Lagrange coefficients $l_i,\ 0 \leq i \leq 19$, see Figure 8.1. A method to approximate a Hex20 serendipity element $F(r,s,t)$ by a set of linear-edge quadratic tetrahedra is discussed. Once converted, the visualization techniques described in the previous chapters can be utilized to visualize the Hex20 element. The Hex20 element is defined as

$$F(r,s,t) = \sum_{0}^{19} l_i \mathcal{L}_i(r,s,t),\ -1 \leq r,s,t \leq 1, \tag{8.1}$$

where the Lagrange basis (or shape) functions $\mathcal{L}_i$ associated with each knot i of a Hex20

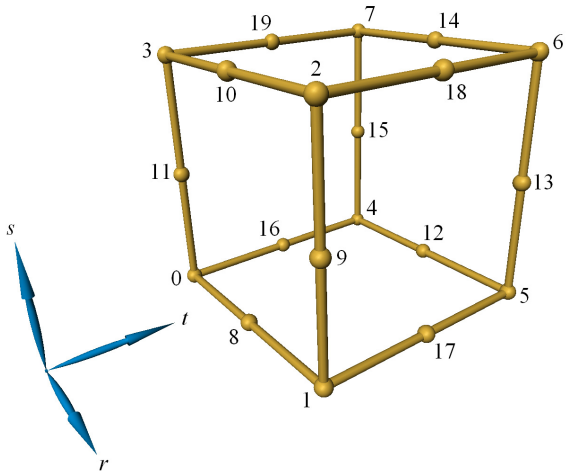

Figure 8.1: Lagrange coefficient labelling used for serendipity quadratic hexahedron (Hex20).

element are given as

$$
\begin{aligned}
\mathcal{L}_0 &= \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2), \\
\mathcal{L}_1 &= \frac{1}{8}(1+r)(1-s)(1-t)(r-s-t-2), \\
\mathcal{L}_2 &= \frac{1}{8}(1+r)(1+s)(1-t)(r+s-t-2), \\
\mathcal{L}_3 &= \frac{1}{8}(1-r)(1+s)(1-t)(-r+s-t-2), \\
\mathcal{L}_4 &= \frac{1}{8}(1-r)(1-s)(1+t)(-r-s+t-2), \\
\mathcal{L}_5 &= \frac{1}{8}(1+r)(1-s)(1+t)(r-s+t-2), \\
\mathcal{L}_6 &= \frac{1}{8}(1+r)(1+s)(1+t)(r+s+t-2), \\
\mathcal{L}_7 &= \frac{1}{8}(1-r)(1+s)(1+t)(-r+s+t-2), \\
\mathcal{L}_8 &= \frac{1}{4}(1-r^2)(1-s)(1-t), \\
\mathcal{L}_9 &= \frac{1}{4}(1-s^2)(1+r)(1-t), \\
\mathcal{L}_{10} &= \frac{1}{4}(1-r^2)(1+s)(1-t), \\
\mathcal{L}_{11} &= \frac{1}{4}(1-s^2)(1-r)(1-t), \\
\mathcal{L}_{12} &= \frac{1}{4}(1-r^2)(1-s)(1+t), \\
\mathcal{L}_{13} &= \frac{1}{4}(1-s^2)(1+r)(1+t), \\
\mathcal{L}_{14} &= \frac{1}{4}(1-r^2)(1+s)(1+t), \\
\mathcal{L}_{15} &= \frac{1}{4}(1-s^2)(1-r)(1+t), \\
\mathcal{L}_{16} &= \frac{1}{4}(1-t^2)(1-r)(1-s), \\
\mathcal{L}_{17} &= \frac{1}{4}(1-t^2)(1+r)(1-s), \\
\mathcal{L}_{18} &= \frac{1}{4}(1-t^2)(1+r)(1+s), \text{ and} \\
\mathcal{L}_{19} &= \frac{1}{4}(1-t^2)(1-r)(1+s). \qquad (8.2)
\end{aligned}
$$

Five quadratic tetrahedra are used to approximate a Hex20 element. Figure 8.2 shows the knot labelling for the quadratic tetrahedra. The indices of the five tetrahedra are given by (0, 1, 3, 4, 8, 11, 16, 20, 22, 25), (5, 4, 6, 1, 12, 13, 17, 21, 22, 23), (2, 3, 1, 6, 10, 9, 18, 20, 24, 23), (7, 6, 4, 3, 14, 15, 19, 21, 24, 25), and (3, 1, 6, 4, 20, 24, 25, 23, 22, 21), where—for a tetrahedron with indices (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)—the first four indices are

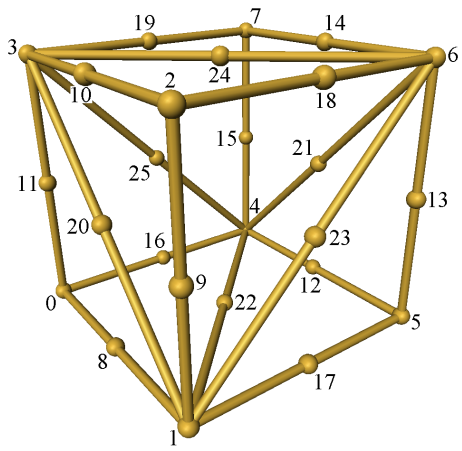

Figure 8.2: Vertex labelling used for five-tetrahedra approximation of Hex20 element.

the corner knots and the remaining six are the edge knots between knot pairs (0,1), (0,2), (0,3), (1,2), (1,3), and (2,3), respectively. The five-tetrahedra approximation introduces additional knots on the faces of the hexahedral element. The coefficients for these knots are computed by fitting a quadratic curve to the three values obtained by evaluating $F(r, s, t)$ at the endpoints and midpoint defining the edge containing the new knot. The coefficients c_i used by the tetrahedra are given by

$$
\begin{aligned}
c_0 &= l_0, \\
c_1 &= l_1, \\
c_2 &= l_2, \\
c_3 &= l_3, \\
c_4 &= l_4, \\
c_5 &= l_5, \\
c_6 &= l_6, \\
c_7 &= l_7, \\
c_8 &= 2l_8 - \frac{l_0 + l_1}{2}, \\
c_9 &= 2l_9 - \frac{l_1 + l_2}{2}, \\
c_{10} &= 2l_{10} - \frac{l_2 + l_3}{2}, \\
c_{11} &= 2l_{11} - \frac{l_3 + l_0}{2},
\end{aligned}
$$

$$
\begin{aligned}
c_{12} &= 2l_{12} - \frac{l_4 + l_5}{2}, \\
c_{13} &= 2l_{13} - \frac{l_5 + l_6}{2}, \\
c_{14} &= 2l_{14} - \frac{l_6 + l_7}{2}, \\
c_{15} &= 2l_{15} - \frac{l_7 + l_4}{2}, \\
c_{16} &= 2l_{16} - \frac{l_0 + l_4}{2}, \\
c_{17} &= 2l_{17} - \frac{l_1 + l_5}{2}, \\
c_{18} &= 2l_{18} - \frac{l_2 + l_6}{2}, \\
c_{19} &= 2l_{19} - \frac{l_3 + l_7}{2}, \\
c_{20} &= l_8 + l_9 + l_{10} + l_{11} - l_1 - l_3 - \frac{l_0 + l_2}{2}, \\
c_{21} &= l_{12} + l_{13} + l_{14} + l_{15} - l_4 - l_6 - \frac{l_5 + l_7}{2}, \\
c_{22} &= l_8 + l_{12} + l_{16} + l_{17} - l_1 - l_4 - \frac{l_0 + l_5}{2}, \\
c_{23} &= l_9 + l_{13} + l_{17} + l_{18} - l_1 - l_6 - \frac{l_2 + l_5}{2}, \\
c_{24} &= l_{10} + l_{14} + l_{18} + l_{19} - l_3 - l_6 - \frac{l_2 + l_7}{2}, \text{ and} \\
c_{25} &= l_{11} + l_{15} + l_{16} + l_{19} - l_3 - l_4 - \frac{l_0 + l_7}{2}.
\end{aligned} \tag{8.3}
$$

The five-tetrahedra approximation can represent $F(r, s, t)$ exactly along the edges of the Hex20, however, the values over the faces and the interior are approximated. The error e_i of the i^{th} tetrahedron T_i in the approximation is estimated using the root-mean-square (RMS) error over the domain of T_i, defined as

$$
e_i = \sqrt{\int_{T_i} (F - T_i)^2}, \tag{8.4}
$$

where F is the Hex20 element being approximated. (This error is computed analytically, since F and T_i are known analytically.) To evaluate the quality of the approximation, the approximation error for several random hexahedral elements was computed. Each hexahedron was defined over the domain of the unit cube and Lagrange coefficients were randomly generated in the range $[0, 1]$. The average RMS error e_i for each of the five tetrahedra

$T_i,\ 0 \leq i \leq 4$, for 1000000 individual random hexahedral elements is

$$\begin{aligned} e_0 &= 6.253e^{-2}, \\ e_1 &= 6.255e^{-2}, \\ e_2 &= 6.255e^{-2}, \\ e_3 &= 6.247e^{-2}, \text{ and} \\ e_4 &= 7.262e^{-2}. \end{aligned} \tag{8.5}$$

(Tetrahedron T_4 has a higher error e_4 than the other tetrahedra, since it is larger in volume.) When approximating a grid of Hex20 elements, one must alternate the indexing, so that neighboring elements share edges correctly.

Chapter 9

Conclusions and Future Work

Higher-order elements reduce the required number of elements needed to represent data significantly. Efficient means to use them, whether for approximation or visualization, should be studied. Consider the scenario of having to send a representation of a large data set across a network. One could send several linear elements or a few higher-order elements, saving on bandwidth, and speeding up the transfer. This speed-up permits interactive visualization of relatively large data sets on relatively more inexpensive machines.

The methods for approximation, isosurfacing, ray casting, and cutting planes are meant to be foundations for future research to extend and optimize upon. With video hardware making advances in support of higher-order patch rendering and programmable vertex and pixel shaders, hardware-assisted visualization of higher-order elements is feasible and has the potential to be competitive with conventional linear techniques. Usage of higher-order elements will succeed when three problems are solved:

1. Creating curved-element decompositions of domains.
2. Obtaining C^1-continuous (or G^1-continuous) isosurface representations.
3. Improving performance of approximation and visualization.

Each problem is discussed:

9.1 Curved Domain Decomposition

Curved elements can better decompose domains, since element edges can be aligned with features (and boundaries) in the underlying data. This, in addition to better field approximation, further reduces the required number of elements and provides a better representation of the data. Existing curvilinear grids—that already take advantage of such features—may be easily converted to use curved elements, however, non-curvilinear data (i.e., rectilinear grids and unstructured meshes) cannot benefit so easily. Methods to construct artificial features and align curved elements along those features are needed. For example, it may be reasonable to extract several isosurfaces from a rectilinear data set, fit higher-order patches to them, and then triangulate the data in-between patches to form a curved-quadratic tetrahedral representation of the data, see [48]. Alternatively, scientists could use curved-element meshes from the beginning.

9.2 Isosurface Continuity

It is visually (and possibly functionally) important for an isosurface to appear "smooth" to a viewer. Using the isosurfacing method described, an isosurface from a higher-order element results in C^0-continuous patches. This tends to be quite obvious if the surface is extracted from a coarse mesh. A reasonable solution is to smooth the resulting patches, so that they appear to be C^1-continuous. This can probably be done by first raising the degree of the resulting quadratic patches to a quartic (an isosurface from a curved-quadratic element already uses a quartic representation, thus, no elevation in degree is required). Then, the three control points in the interior of each patch can be adjusted so that neighboring elements share tangent planes across each edge, making them G^1-continuous (tangent-plane-continuous). (It may also be possible to make them C^1-continuous using a variation of this method.)

9.3 Performance

To be competitive with linear elements, higher-order element speed, in general, must be addressed. This concerns the generation of approximations and visualizations of higher-order elements. In terms of approximation, the method described in this dissertation is slow for both linear and higher-order elements. This behavior is caused by having to sample the function F being approximated several times at arbitrary locations. Better integration and sampling strategies are needed to accelerate this required step. (Acceleration would greatly improve performance of linear approximation techniques as well.)

Regarding the speed of visualization, none of the methods described in this dissertation were optimized. Many research possibilities exist for the optimization and improvement of the described methods. Assisting any of these methods with hardware is desirable and, since higher-order patch rendering is already supported, feasible.

Bibliography

[1] C.L. Bajaj, *Free-form modeling with implicit surface patches*, Implicit Surfaces, J. Bloomenthal and B. Wyvill (Eds.), Morgan Kaufman Publishers, San Francisco, CA, 1996

[2] R.E. Barnhill and F.F. Little, *Adaptive Triangular Cubatures*, Rocky Mountain J. Math., 14, 1, pp. 53–75, 1984

[3] B.K. Bloomquist, *Contouring Trivariate Surfaces*, Masters Thesis, Arizona State University, Computer Science Department, Tempe, AZ, 1990

[4] W. Boehm and H. Prautzsch, *Numerical Methods*, Wellesley, Massachussets, A.K. Peters, Ltd., 1993

[5] G.P. Bonneau, S. Hahmann, and G.M. Nielson, *BLAC-wavelets: A multiresolution analysis with non-nested spaces*, in: Yagel, R. and Nielson, G. M. (Eds.) Visualization '96, IEEE Computer Society Press, Los Alamitos, CA, pp. 43–48, 1996

[6] A. Cantoni, *Optimal curve fitting with piecewise linear functions*, IEEE Trans. Computers, 20(1), pp. 59–67, 1971

[7] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno, *Multiresolution modeling and visualization of volume data based on simplicial complexes*, in: A.E. Kaufman and W. Krüger (Eds.), 1994 Symposium on Volume Visualization, IEEE Computer Society Press, Los Alamitos, CA, pp. 19–26, 1994

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

[9] R.D. Cook, D.S. Malkus, and M.E. Plesha, *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, New York, 1989

[10] P.J. Davis, *Interpolation and Approximation*, New York, Dover, 1975

[11] N. Dyn, M.S. Floater, and A. Iske, *Adaptive thinning for bivariate scattered data*, Technische Universität München, Fakultät für Mathematik, München, Germany, Report TUM M0006, 2000.

[12] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, fifth edition, Academic Press, San Diego, CA., 2002

[13] M.S. Floater and A. Iske, *Thinning and approximation of large sets of scattered data*, in: Fontanella, F., Jetter K. and Laurent, P. J. (Eds.), Advanced Topics in Multivariate Approximation, World Scientific, Singapore, pp. 87-96, 1996

[14] R. Franke, *Scattered data interpolation: Tests of some methods*, Math. Comp. 38, pp. 181–200, 1982

[15] M.H. Gross, R. Gatti, and O.G. Staadt, *Fast multiresolution surface meshing*, in: G.M. Nielson and D. Silver (Eds.), Visualization '95, IEEE Computer Society Press, Los Alamitos, CA, pp. 135–142, 1995

[16] T.S. Gieng, B. Hamann, K.I. Joy, G.L. Schussman, and I.J. Trotts, *Smooth hierarchical surface triangulations*, in: R. Yagel and H. Hagen (Eds.), Visualization '97, IEEE Computer Society Press, Los Alamitos, CA, pp. 379–386, 1997

[17] T.S. Gieng, B. Hamann, K.I. Joy, G.L. Schussman, and I.J. Trotts, *Constructing hierarchies for triangle meshes*, IEEE Transactions on Visualization and Computer Graphics 4(2), pp. 145–161, 1998

[18] B.F. Gregorski, M.A. Duchaineau, P. Lindstrom, V. Pascucci, and K.I. Joy, *Interactive view-dependent rendering of large isosurfaces*, in: Proceedings of IEEE Visualization 2002, R. Moorhead, M. Gross, and K.I. Joy (Eds.), pp. 475-482, 2002

[19] R. Grosso, C. Lürig, and T. Ertl, *The multilevel finite element method for adaptive mesh optimization and visualization of volume data*, in: R. Yagel and H. Hagen (Eds.), Visualization '97, IEEE Computer Society Press, Los Alamitos, CA, pp. 387–394, 1997

[20] B. Hamann, *A data reduction scheme for triangulated surfaces*, Computer Aided Geometric Design 11(2), Elsevier, pp. 197–214, 1994

[21] B. Hamann, *Modeling contours of trivariate data*, Mathematical Modelling and Numerical Analysis (Modelisation Mathematique et Analysis Numerique) 26(1), Gauthier-Villars, France, pp. 51–75, 1992

[22] B. Hamann, *Visualization and modeling contours of trivariate functions*, Ph.D. dissertation, Department of Computer Science, Arizona State University, Tempe, Arizona, 1991

[23] B. Hamann, B.W. Jordan, and D.F. Wiley, *On a construction of a hierarchy of best linear spline approximations using repeated bisection*, IEEE Transactions on Visualization and Computer Graphics 5(1), pp. 30–46, and 5(2), 190 (errata), 1999

[24] B. Hamann, I.J. Trotts, and G. Farin, *On approximating contours of the piecewise trilinear interpolant using triangular rational-quadratic Bézier patches*, IEEE Transactions on Visualization and Computer Graphics, 3(3), pp. 315– 337, 1997

[25] G. Hämmerlin and K.H. Hoffmann, *Numerical Math*, New York, Springer-Verlag, 1991

[26] P.S. Heckbert and M. Garland, *Survey of polygonal surface simplification algorithms*, Technical Report, Computer Science Department, Carnegie Mellon University, Pittsburg, Pennsylvania, 1997

[27] H. Hoppe, *Progressive meshes*, in: H. Rushmeier (Ed.), Proceedings of SIGGRAPH 1996, ACM Press, New York, NY, pp. 99–108, 1996

[28] H. Hoppe, *View-dependent refinement of progressive meshes*, in: T. Whitted (Ed.), Proceedings of SIGGRAPH 1997, ACM Press, New York, NY, pp. 189–198, 1997

[29] A. Iserles, P.E. Koch, S.P. Nørsett, and J.M. Sanz-Serna, *Orthogonality and approximation in a Sobolev space*, in: J.C. Mason and M.G. Cox (Eds.), Algorithms for Approximation II, Chapman and Hall, London, pp. 117–124, 1990

[30] A.E. Kaufman, *Volume Visualization*, IEEE Computer Society Press, Los Alamitos, CA, 1991

[31] R. Khardekar and D. Thompson, *Rendering higher order finite element surfaces in hardware*, in Proceedings of GRAPHITE 2003, M. Adcock, I. Gwilt, and L. Y. Tsui (Eds.), ACM, Feb 11-14, Melbourne, Australia, pp. 211–ff, 2003

[32] E. Lee, *The rational Bézier representation for conics*, in: G. Farin (Ed.), Geometric Modeling, SIAM, Philadelphia, PA, 1987

[33] W.E. Lorenson and H.E. Cline, *Marching cubes: a high resolution 3D surface construction algorithm*, Computer Graphics, 21, pp. 163–169, 1987

[34] S. Marlow and M.J.D. Powell, *A Fortran subroutine for plotting the part of a conic that is inside a given triangle*, Report no. R 8336, Atomic Energy Research Establishment, Harwell, United Kingdom, 1976

[35] N. Max, P. Williams, and C. Silva, *Cell projection of meshes with non-planar faces*, in "Scientific Visualization, Dagstuhl 2000," 2002

[36] G.M. Nielson, *Scattered data modeling*, IEEE Computer Graphics and Applications 13(1), pp. 60–70, 1993

[37] S. Parker, M. Parker, Y. Livnat, P.P. Sloan, C. Hansen, and P. Shirley, *Interactive ray tracing for volume visualization*, IEEE Transactions on Visualization and Computer Graphics, 5(3), pp. 238–250, 1999

[38] T. Petersdorff, *A short proof for Romberg integration*, in: The American Mathematical Monthly, 100(8), pp. 783–785, October, 1993

[39] S. Rippa, *Long and thin triangles can be good for linear interpolation*, SIAM J. Numer. Anal. 29(1), pp. 257–270, 1992

[40] O.G. Staadt, M.H. Gross, and R. Weber, *Multiresolution compression and reconstruction*, in: R. Yagel and H. Hagen (Eds.), Visualization '97, IEEE Computer Society Press, Los Alamitos, CA, pp. 337–346, 1997

[41] H. Stone, *Approximation of curves by line segments*, Math. Computation, 15, pp. 40–47, 1961

[42] I. Tomek, *Two algorithms for piecewise-linear continuous approximation of functions of one variable*, IEEE Trans. Computers, 23, pp. 45–48, 1974

[43] A. Vlachos, J. Peters, C. Boyd and J.L. Mitchell, *Curved PN Triangles*, ACM Symposium on Interactive 3D Graphics 2001, pp. 159–166, 2001

[44] D.F. Wiley, M. Bertram, and B. Hamann, *On a construction of a hierarchy of best linear spline approximations using a finite element approach*, IEEE Transactions on Visualization and Computer Graphics, to appear, 2003

[45] D.F. Wiley, M. Bertram, B. Hamann, K.I. Joy, N.L. Max, and G. Scheuermann, *Hierarchical spline approximation*, in: G. Farin, B. Hamann, and H. Hagen (Eds.), Hierarchical and Geometrical Methods in Scientific Visualization, Springer-Verlag, Heidelberg, Germany, pp. 63–88, 2003

[46] D.F. Wiley, H.R. Childs, B.F. Gregorski, B. Hamann, and K.I. Joy, *Contouring curved quadratic elements*, in: G.P. Bonneau, S. Hahmann, and C.D. Hansen (Eds.), Data Visualization 2003, Proceedings of VisSym 2003, 2003

[47] D.F. Wiley, H.R. Childs, B. Hamann, K.I. Joy, and N.L. Max, *Best quadratic spline approximation for hierarchical visualization*, in: D. Ebert, P. Brunet, and I. Navazo (Eds.), Data Visualization 2002, Proceedings of VisSym 2002, pp. 133–140, 2002

[48] D.F. Wiley, H.R. Childs, B. Hamann, K.I. Joy, and N.L. Max, *Using quadratic simplicial elements for hierarchical approximation and visualization* , in: R.F. Erbacher, P.C. Chen, M. Groehn, J.C. Roberts, and C.M. Wittenbrink (Eds.), Visualization and Data Analysis 2002, Proceedings, pp. 32–43, 2002

[49] P.L. Williams, N.L. Max, and C.M. Stein, *A high accuracy volume renderer for unstructured data*, IEEE Transactions on Visualization and Computer Graphics, 4(1), pp. 37–54, 1998

[50] A.J. Worsey and G. Farin, *Contouring a bivariate quadratic polynomial over a triangle*, Computer Aided Geometric Design 7 (1–4), pp. 337–352, 1990

Appendix A

Romberg Integration

It is non-trivial to perform integration of an arbitrary function defined over intervals, triangles, and tetrahedra. A method that extends easily from 1D to 3D is Romberg integration and is discussed in the following sections.

A.1 The 1D Case

For the integration of a function $f(x)$ over the unit interval $[0, 1]$ one computes a sequence of trapezoidal sums,

$$A_t = \frac{1}{2^t}\frac{1}{2}\left(f_0 + f_{2^t} + 2\sum_{i=1}^{2^t-1} f_i\right), \quad t = 0, \ \ldots, \ n, \tag{A.1}$$

where $f_i = f(\frac{i}{2^t})$, and uses this sequence to compute extrapolated, usually better, integral approximations. The sequence of A_t values converges linearly to the exact value of the integral of $f(x)$ defined over the unit interval, see [38].

Having computed the values $A_0^0 = A_0$, $A_1^0 = A_1$, $A_2^0 = A_2$, $\ldots$, $A_n^0 = A_n$, the extrapolated approximation is

$$A_i^j = \frac{A_{i+1}^{j-1} - 2^{-2j}A_i^{j-1}}{1 - 2^{-2j}}, \ j = 1, \ \ldots, \ n, \quad i = 0, \ \ldots, \ n - j. \tag{A.2}$$

Initially, the triangular Romberg scheme is computed for $n = 1$ and then n is increased one-by-one until the condition $|A_0^n - A_1^n| < \epsilon$, where ϵ is some user-defined tolerance. Figure A.1 illustrates the triangular Romberg scheme resulting from Equation (A.2).

$$
\begin{array}{ccccccccccc}
A_0^0 & \rightarrow & A_0^1 & \rightarrow & A_0^2 & \dots & A_0^{n-2} & \rightarrow & A_0^{n-1} & \rightarrow & A_0^n \\
A_1^0 & \nearrow & A_1^1 & \nearrow & \vdots & & A_1^{n-2} & \nearrow & A_1^{n-1} & \nearrow & \\
A_2^0 & \nearrow & \vdots & \nearrow & \vdots & & A_2^{n-2} & \nearrow & & & \\
\vdots & \nearrow & \vdots & \nearrow & A_{n-2}^2 & & & & & & \\
\vdots & \nearrow & A_{n-1}^1 & \nearrow & & & & & & & \\
A_n^0 & \nearrow & & & & & & & & &
\end{array}
$$

Figure A.1: Triangular Romberg scheme.

A.2 The 2D Case

Regarding the computation of scalar products $\langle F, f_i \rangle$ and error estimates in the bivariate case, one can choose from a large pool of numerical integration schemes. A simple yet robust and efficient adaptive triangular cubature scheme is described in [2] and one could use it as an alternative to the bivariate Romberg scheme described here. The Romberg scheme was chosen, since it generalizes nicely to arbitrary dimension. Regardless of the chosen numerical method, all cubature methods assume that one knows an analytical definition of the function being integrated—this can be a discontinuous, C^0-, C^1-, ..., or C^∞-continuous definition, and that one can effectively evaluate the function. The bivariate Romberg scheme and the construction of a sequence of integral estimates based on linear-edge triangular elements, is briefly discussed. The description is limited to the standard triangle—having vertices $\mathbf{v}_{0,0}^0 = (0,0)^{\mathrm{T}}$, $\mathbf{v}_{1,0}^0 = (1,0)^{\mathrm{T}}$, and $\mathbf{v}_{0,1}^0 = (0,1)^{\mathrm{T}}$.

An initial estimate of the value of $\int f(x,y)dxdy$ for some function f defined over the standard triangle is

$$A_0 = \frac{1}{2}\frac{1}{3}\Big(f(0,0) + f(1,0) + f(0,1)\Big), \tag{A.3}$$

which is the area of the standard triangle multiplied by the average of the three function values at the three vertices. One obtains a better estimate by splitting the standard triangle into four subtriangles, and adding the resulting values. The six vertices defining the vertices of the four subtriangles are $\mathbf{v}_{0,0}^1 = (0,0)^{\mathrm{T}}$, $\mathbf{v}_{1,0}^1 = (\frac{1}{2},0)^{\mathrm{T}}$, $\mathbf{v}_{2,0}^1 = (1,0)^{\mathrm{T}}$, $\mathbf{v}_{0,1}^1 = (0,\frac{1}{2})^{\mathrm{T}}$, $\mathbf{v}_{1,1}^1 = (\frac{1}{2},\frac{1}{2})^{\mathrm{T}}$, and $\mathbf{v}_{0,2}^1 = (0,1)^{\mathrm{T}}$, see Figure A.2. The vertex triples defining the four sub-

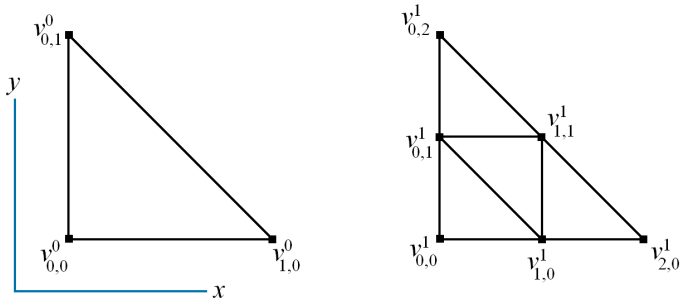

Figure A.2: Indexing of first and second levels in Romberg integration scheme.

triangles are $(\mathbf{v}^1_{0,0}, \mathbf{v}^1_{1,0}, \mathbf{v}^1_{0,1})$, $(\mathbf{v}^1_{1,0}, \mathbf{v}^1_{2,0}, \mathbf{v}^1_{1,1})$, $(\mathbf{v}^1_{0,1}, \mathbf{v}^1_{1,1}, \mathbf{v}^1_{0,2})$, and $(\mathbf{v}^1_{1,0}, \mathbf{v}^1_{1,1}, \mathbf{v}^1_{0,1})$. Multiplying the areas of the subtriangles with the averages of the three function values at their respective vertices and adding the individual results yields the approximation

$$A_1 = \frac{1}{2^2}\frac{1}{2}\frac{1}{3}\left(f_{0,0} + f_{2,0} + f_{0,2} + 3f_{1,0} + 3f_{0,1} + 3f_{1,1}+\right), \tag{A.4}$$

where $f_{i,j} = f(\mathbf{v}^1_{i,j})$. The general level-$t$ approximation is given by

$$\begin{aligned} A_t \;=\; & \tfrac{1}{2^{2t}}\tfrac{1}{2}\tfrac{1}{3}\Big(f_{0,0} + f_{2^t,0} + f_{0,2^t} + 3\textstyle\sum_{i=1}^{2^t-1} f_{i,0} + 3\sum_{j=1}^{2^t-1} f_{0,j} + \\ & 3\textstyle\sum_{i,j>0,\ i+j=2^t} f_{i,j} + 6\sum_{j=1}^{2^t-2}\sum_{i=1}^{2^t-1-j} f(i,j)\Big) \end{aligned}, \tag{A.5}$$

where $f_{i,j} = f\left(\frac{i}{2^t}, \frac{j}{2^t}\right)$. This is a generalization of the trapezoidal sums used in the 1D case, see [38]. The A_t values are then used to compute integral approximations A^j_i using the triangular Romberg scheme.

A.3 The 3D Case

Romberg integration is used for the computation of scalar products $\langle F, f_i\rangle$ and error values. The description is limited to the standard tetrahedron. Subdivision of the standard tetrahedron into subpolyhedra is more complicated than subdividing the standard triangle. An initial estimate of $\int f(x,y,z)dxdydz$ for some function f defined over the standard tetrahedron is

$$A_0 = \frac{1}{6}\frac{1}{4}\Big(f(0,0,0) + f(1,0,0) + f(0,1,0) + f(0,0,1)\Big), \tag{A.6}$$

which is the volume of the standard tetrahedron—having vertices $\mathbf{v}^0_{0,0,0} = (0,0,0)^{\mathrm{T}}$, $\mathbf{v}^0_{1,0,0} = (1,0,0)^{\mathrm{T}}$, $\mathbf{v}^0_{0,1,0} = (0,1,0)^{\mathrm{T}}$, and $\mathbf{v}^0_{0,0,1} = (0,0,1)^{\mathrm{T}}$—multiplied by the average of the four function values at the four vertices. One obtains a better estimate of the integral by decomposing the standard tetrahedron into four tetrahedra and one octahedron (which is split into four subtetrahedra), estimating integral values for the four tetrahedra and the octahedron, and adding the individual results. The 10 vertices defining the first-level decomposition are $\mathbf{v}^1_{i,j,k} = \left(\frac{i}{2}, \frac{j}{2}, \frac{k}{2}\right)^{\mathrm{T}}$, $k = 0, \ \ldots, \ 2, \ j = 0, \ \ldots, \ 2-k, \ i = 0, \ \ldots, \ 2-k-j$. The vertex quadruples defining the four tetrahedra are $(\mathbf{v}^1_{0,0,0}, \mathbf{v}^1_{1,0,0}, \mathbf{v}^1_{0,1,0}, \mathbf{v}^1_{0,0,1})$, $(\mathbf{v}^1_{1,0,0}, \mathbf{v}^1_{2,0,0}, \mathbf{v}^1_{1,1,0}, \mathbf{v}^1_{1,0,1})$, $(\mathbf{v}^1_{0,1,0}, \mathbf{v}^1_{1,1,0}, \mathbf{v}^1_{0,2,0}, \mathbf{v}^1_{0,1,1})$, and $(\mathbf{v}^1_{0,0,1}, \mathbf{v}^1_{1,0,1}, \mathbf{v}^1_{0,1,1}, \mathbf{v}^1_{0,0,2})$. The vertex tuple $(\mathbf{v}^1_{1,0,0}, \mathbf{v}^1_{1,1,0}, \mathbf{v}^1_{0,1,0}, \mathbf{v}^1_{0,0,1}, \mathbf{v}^1_{1,0,1}, \mathbf{v}^1_{0,1,1})$ defines the octahedron. The octahedron is split into four subtetrahedra of equal volume by adding an edge connecting $\mathbf{v}^1_{0,0,1}$ and $\mathbf{v}^1_{1,1,0}$. Multiplying the volumes of the tetrahedra and subtetrahedra with the average of the four function values at their respective vertices and adding the individual results yields the approximation

$$\begin{aligned} A_1 \ = \ & \tfrac{1}{2^3}\tfrac{1}{6}\tfrac{1}{4}\left(\textstyle\sum_{k=0}^{1}\sum_{j=0}^{1-k}\sum_{i=0}^{1-k-j}\left(f_{i,j,k} + f_{i+1,j,k} + f_{i,j+1,k} + f_{i,j,k+1}\right) + \right. \\ & \left. 4f_{0,0,1} + 4f_{1,1,0} + 2f_{1,0,0} + 2f_{0,1,0} + 2f_{1,0,1} + 2f_{0,1,1}\right), \end{aligned} \tag{A.7}$$

where $f_{i,j,k} = f(\mathbf{v}^1_{i,j,k})$. The general level-$t$ approximation is given by

$$\begin{aligned} A_t \ = \ & \tfrac{1}{2^{3t}}\tfrac{1}{6}\tfrac{1}{4}\left(\textstyle\sum_{k=0}^{2^t-1}\sum_{j=0}^{2^t-1-k}\sum_{i=0}^{2^t-1-k-j}\left(f_{i,j,k} + f_{i+1,j,k} + f_{i,j+1,k} + f_{i,j,k+1}\right) + \right. \\ & \textstyle\sum_{k=0}^{2^t-2}\sum_{j=0}^{2^t-2-k}\sum_{i=0}^{2^t-2-k-j}\left(4f_{i,j,k+1} + 4f_{i+1,j+1,k} + 2f_{i+1,j,k} + \right. \\ & \left. 2f_{i,j+1,k} + 2f_{i+1,j,k+1} + 2f_{i,j+1,k+1}\right) + \\ & \textstyle\sum_{k=0}^{2^t-3}\sum_{j=0}^{2^t-3-k}\sum_{i=0}^{2^t-3-k-j}\left(f_{i+1,j+1,k} + f_{i+1,j,k+1} + \right. \\ & \left. \left. f_{i,j+1,k+1} + f_{i+1,j+1,k+1}\right)\right), \end{aligned} \tag{A.8}$$

where $f_{i,j,k} = f\left(\frac{i}{2^t}, \frac{j}{2^t}, \frac{k}{2^t}\right)$. This is a generalization of the trapezoidal sums used in the 1D case. The A_t values are then used to compute integral approximations A^j_i using the triangular Romberg scheme.

Appendix B

Image-space Error

In some applications, it is more appropriate to compare resulting images (from a visualization) of an approximation rather than to compute an analytical error measurement for the approximation. To do this, a difference image is constructed between an "original" image and an image of the approximation. A quantitative measurement, the *image-space error*, is computed by integrating over the difference image. For an $M \times N$ RGB pixel image, the image-space error E (as a percentage) is given by

$$E = \left(\frac{\sum_{j=0}^{N-1} \sum_{i=0}^{M-1} I_{i,j}}{MNV} \right) 100, \tag{B.1}$$

where $I_{i,j} = \frac{r_{i,j}+g_{i,j}+b_{i,j}}{3}$ is the intensity of the pixel at $(i,j)^{\mathrm{T}}$ and V is the maximum value that an intensity $I_{i,j}$ can obtain (for eight-bit RGB data $V = 255$). Thus, if two images are exactly the same, the difference image is "black" and the error is 0%. Two images that are exactly opposite produce a "white" image with an error of 100%. This is more intuitive than an arbitrary RMS error measurement. Figures B.1 and B.2 show difference images—and their associated image-space errors—formed from two sets of images.

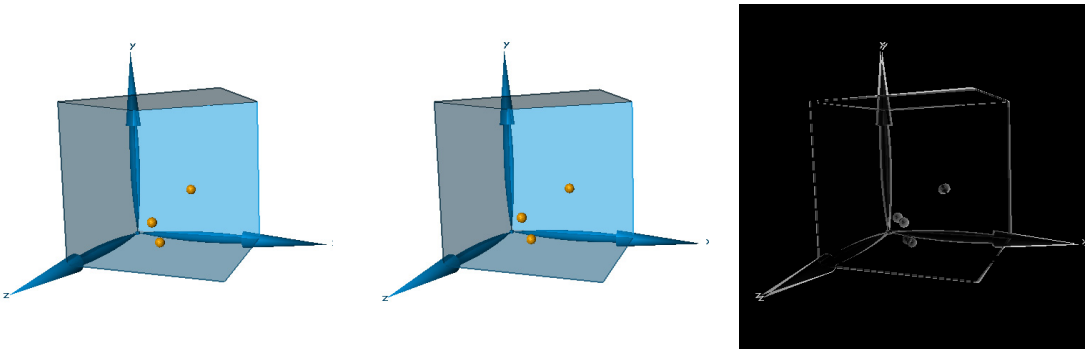

Figure B.1: Image-space error measurement. Left and middle images show images being compared. Right image shows the difference image (error E being 1.35%).

Figure B.2: Image-space error measurement. Left image shows the original. Middle image shows the approximation. Right image shows the difference image (error E being 3.56%).

Appendix C

Mapping Control Net

The similarities between the control net of $\mathbf{Q}(\mathbf{u})$ and that of $\mathbf{C}(\mathbf{u})$ are proved, as illustrated in Figure 5.6. The property for the left tangent line formed by $\mathbf{p}_0$ and $\mathbf{p}_1$ is proved first, where $\mathbf{p}_i = (u_i, v_i)^{\mathrm{T}},\ 0 \leq i \leq 2$. It must be shown that $\mathbf{l}_0 = \mathbf{d}_0$ and $\mathbf{l}_1 = \mathbf{d}_1$. First, it is found that $\mathbf{l}_0 = \mathbf{T}(\mathbf{p}_0)$ and $\mathbf{l}_2 = \mathbf{T}(\mathbf{p}_1)$. The variable $\mathbf{l}_0$ is given as

$$\mathbf{l}_0 = \left[\begin{bmatrix} \mathbf{b}_{20} & \mathbf{b}_{11} & \mathbf{b}_{10} \\ \mathbf{b}_{11} & \mathbf{b}_{02} & \mathbf{b}_{01} \\ \mathbf{b}_{10} & \mathbf{b}_{01} & \mathbf{b}_{00} \end{bmatrix} \begin{bmatrix} u_0 \\ v_0 \\ 1-u_0-v_0 \end{bmatrix}\right]^{\mathrm{T}} \begin{bmatrix} u_0 \\ v_0 \\ 1-u_0-v_0 \end{bmatrix}, \tag{C.1}$$

which is the same as

$$\mathbf{l}_0 = \mathbf{c}_0, \tag{C.2}$$

where $\mathbf{c}_0$ is obtained from Equation (5.8), thus, finding that $\mathbf{l}_0 = \mathbf{d}_0$. The variable $\mathbf{l}_2$ is given as

$$\mathbf{l}_2 = \left[\begin{bmatrix} \mathbf{b}_{20} & \mathbf{b}_{11} & \mathbf{b}_{10} \\ \mathbf{b}_{11} & \mathbf{b}_{02} & \mathbf{b}_{01} \\ \mathbf{b}_{10} & \mathbf{b}_{01} & \mathbf{b}_{00} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1-u_1-v_1 \end{bmatrix}\right]^{\mathrm{T}} \begin{bmatrix} u_1 \\ v_1 \\ 1-u_1-v_1 \end{bmatrix}. \tag{C.3}$$

A quadratic curve is fit to $\left\{\mathbf{l}_0, \mathbf{T}\left(\frac{\mathbf{p}_0+\mathbf{p}_1}{2}\right), \mathbf{l}_2\right\}$ and it is found that $\mathbf{l}_1$ is

$$\mathbf{l}_1 = \left[\left[\begin{bmatrix} \mathbf{b}_{00} & 0 & 0 \\ \mathbf{b}_{10}-\mathbf{b}_{00} & \mathbf{b}_{00}+\mathbf{b}_{20}-2\mathbf{b}_{10} & \mathbf{b}_{00}+\mathbf{b}_{11}-\mathbf{b}_{10}-\mathbf{b}_{01} \\ \mathbf{b}_{01}-\mathbf{b}_{00} & \mathbf{b}_{00}+\mathbf{b}_{11}-\mathbf{b}_{10}-\mathbf{b}_{01} & \mathbf{b}_{00}+\mathbf{b}_{02}-2\mathbf{b}_{01} \\ \mathbf{b}_{10}-\mathbf{b}_{00} & 0 & 0 \\ \mathbf{b}_{01}-\mathbf{b}_{00} & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ v_1 \end{bmatrix}\right]^{\mathrm{T}} \begin{bmatrix} 1 \\ u_0 \\ v_0 \\ u_1 \\ v_1 \end{bmatrix}. \tag{C.4}$$

By substituting the solutions for $\mathbf{c}_0$ and $\mathbf{c}_1$, from Equation (5.8), into the solution for $\mathbf{d}_1$ from Equation (5.20) it follows that $\mathbf{d}_1 = \mathbf{l}_1$. A similar proof can be constructed to show that $\mathbf{r}_0 = \mathbf{T}(\mathbf{p}_2) = \mathbf{d}_4$ and $\mathbf{r}_1 = \mathbf{d}_3$, where $\mathbf{r}_1$ is given as

$$\mathbf{r}_1 = \left[\left[\begin{bmatrix} \mathbf{b}_{00} & 0 & 0 \\ \mathbf{b}_{10}-\mathbf{b}_{00} & \mathbf{b}_{00}+\mathbf{b}_{20}-2\mathbf{b}_{10} & \mathbf{b}_{00}+\mathbf{b}_{11}-\mathbf{b}_{10}-\mathbf{b}_{01} \\ \mathbf{b}_{01}-\mathbf{b}_{00} & \mathbf{b}_{00}+\mathbf{b}_{11}-\mathbf{b}_{10}-\mathbf{b}_{01} & \mathbf{b}_{00}+\mathbf{b}_{02}-2\mathbf{b}_{01} \\ \mathbf{b}_{10}-\mathbf{b}_{00} & 0 & 0 \\ \mathbf{b}_{01}-\mathbf{b}_{00} & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ v_1 \end{bmatrix}\right]^{\mathrm{T}} \begin{bmatrix} 1 \\ u_2 \\ v_2 \\ u_1 \\ v_1 \end{bmatrix}. \tag{C.5}$$