Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2004)
O. Deussen, C. Hansen, D.A. Keim, D. Saupe (Editors)

Ray Casting Curved-Quadratic Elements

D. F. Wiley,[1] H. R. Childs,[2] B. Hamann,[1] and K. I. Joy[1]

[1] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science,
University of California, Davis, CA 95616-8562, U.S.A;
email: {wiley, hamann, joy}@cs.ucdavis.edu
[2] Lawrence Livermore National Laboratory, Mail Stop L-098,
7000 East Avenue, Livermore, CA 94550, U.S.A;
email: childs3@llnl.gov

Abstract

We present a method for ray casting curved-quadratic elements in 3D. The advantages of this approach is that a curved element can be directly visualized. Conventionally, higher-order elements are tessellated with several linear elements so that standard visualization techniques can be applied to the linear elements. Our method primarily focuses on how to find an approximation to the intersection between a ray and a curved-quadratic element. Once this approximation is found, conventional accumulation and color mapping techniques can be applied to the approximation to produce a volumetric visualization of the element. A cutting plane implementation is also shown that leverages the ray casting technique.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Raytracing

1. Introduction

Higher-order elements represent more complex regions than linear elements since the overlying polynomial has a higher degree. By having the overlying polynomial bear more of the burden of representing a region, fewer higher-order elements are required—when compared to linear elements—to represent the same region [WCH*02].

The infusion of higher-order elements into mainstream visualization is hindered since they are more difficult to work with than linear elements, not only in terms of visualization, but also in terms of using them in, for example, finite element analysis [CMP89]. Scientists want the advantages of higher-order elements (fewer elements and better quality representation), but they struggle with the problems of more complex mathematics, updating legacy simulation code to use higher-order elements, and creating visualization techniques to view this data.

We note two key properties of a higher-order element. First, the overlying polynomial has a higher order than linear. Second, whether or not the edges of the element are linearly defined. A simple adaption from a linear element to a higher-order element is to only raise the degree of the overlying polynomial. However, it is beneficial to also lift the order of an element's domain so that the edges of the element can be aligned more appropriately with features in the data, i.e., wing geometry in an air flow simulation. These elements (those having higher-order edges) are called *curved higher-order elements*. Figure 1 shows a curved quadratic tetrahedron that has both a quadratically defined overlying polynomial and quadratically defined edges.

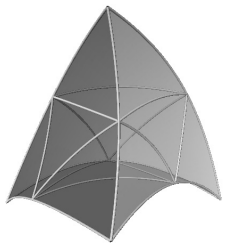

Figure 1: *Example of a curved quadratic tetrahedron having ten control points; one for each corner and one at the midpoint of each edge. (Interior parameter lines, on the faces, are shown to indicate curvature.)*

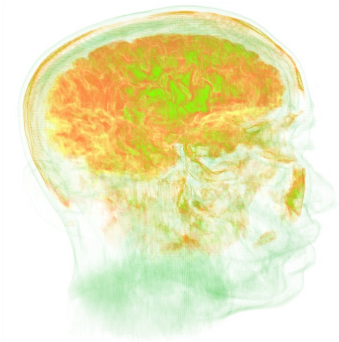

Figure 2: *Ray casting of an MRI (magnetic resonance imaging) data set of human head.*

Some techniques for linear element visualization can be modified slightly and applied to linear-edge higher-order elements. Though in most cases, new techniques for visualization must be developed, for example, extracting isosurfaces directly from quadratic elements [WCG*03]. To further the available tools for visualizing higher-order elements, we present a method for ray casting curved-quadratic elements. (Curved-quadratic elements are elements that have both a quadratically defined polynomial as well as edges.)

A fundamental rendering technique for tetrahedral elements (and all other types of volumetric mesh elements) is ray casting, see [Kau91, PPL*99]. 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 2. 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 of the overlying polynomial. 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.

2. Element Definitions

The ray casting discussion in this paper is focused on quadratic elements, though, the concepts presented can be applied to elements having a higher order than quadratic. A simplicial element in 2D 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 element that represents the dependent variable over the corresponding region in space. Each quadratic basis polynomial is represented in Bernstein-Bézier form, see [Far02].

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

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

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 $T^{\mathbb{U}}$. Thus, a curved-quadratic triangle mapping is defined as

$$T(u,v) = \sum_{j=0}^{2}\sum_{i=0}^{2-j} \mathbf{k}_{i,j} B^2_{i,j}(u,v), \tag{2}$$

where $\mathbf{k}_{i,j}$ are the six control points of the triangle. (We occasionally renumber the knots as $\mathbf{k}_0$, $\mathbf{k}_1$, …, $\mathbf{k}_5$ to correspond to $\mathbf{k}_{0,0}$, $\mathbf{k}_{2,0}$, $\mathbf{k}_{0,2}$, $\mathbf{k}_{1,0}$, $\mathbf{k}_{0,1}$, $\mathbf{k}_{1,1}$, respectively.)

The curved-quadratic tetrahedron is a straightforward extension of the 2D case. A 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

$$B^2_{i,j,k}(u,v,w) = \frac{2!}{(2-i-j-k)!i!j!k!}(1-u-v-w)^{2-i-j-k}u^i v^j w^k, \quad i,j,k \geq 0,\ i+j+k \leq 2, \tag{3}$$

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. Thus, a curved-quadratic tetrahedral mapping is defined as

$$T(u,v,w) = \sum_{k=0}^{2}\sum_{j=0}^{2-k}\sum_{i=0}^{2-k-j} \mathbf{k}_{i,j,k} B^2_{i,j,k}(u,v,w), \tag{4}$$

where $\mathbf{k}_{i,j,k}$ are the ten control points of the tetrahedron. (We occasionally renumber the knots as $\mathbf{k}_0$, $\mathbf{k}_1$, …, $\mathbf{k}_9$ to correspond to $\mathbf{k}_{0,0,0}$, $\mathbf{k}_{2,0,0}$, $\mathbf{k}_{0,2,0}$, $\mathbf{k}_{0,0,2}$, $\mathbf{k}_{1,0,0}$, $\mathbf{k}_{0,1,0}$, $\mathbf{k}_{0,0,1}$, $\mathbf{k}_{1,1,0}$, $\mathbf{k}_{1,0,1}$, $\mathbf{k}_{0,1,1}$, respectively.)

We use two notational variations of the mappings described above. When we refer to $T(\mathbf{u})$ we intend to use the geometrical properties (i.e., physical location of the element in space) of the mapping and when we refer to $T^c(\mathbf{u})$ we are

only considering the coefficient, or functional, properties of the mapping. This is important in the coming sections since we compute and use approximations for both properties independently.

A linear edge mapping $T(\mathbf{u})$ is constructed for an element by using the standard linear mapping of a linear triangle and tetrahedron having three and four control points, respectively. The functional mapping $T^c(\mathbf{u})$, however, still uses the quadratic definition as described by Equations (2) and (4). This combination of linear-physical and quadratic-functional mapping is called a linear-edge quadratic element.

3. Ray Casting Overview

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 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 defined 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. Thus, two approximations $C(\mathbf{u})$ and $C^c(\mathbf{u})$ are constructed. $C(\mathbf{u})$ represents the intersection segment in parameter space and $C^c(\mathbf{u})$ represents the functional over that segment. Once $C^c(\mathbf{u})$ is computed, $C(\mathbf{u})$ is discarded since only the functional information is needed along the ray.

4. 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.

4.1. 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 $c_i,\ 0 \le i \le 2$, distributed uniformly across its domain $t \in [0,1]$, is defined as

$$C^c(t) = \sum_{i=0}^{2} c_i B_i^2(t), \tag{5}$$

where B_i^2 is the univariate n^{th}-degree Bernstein polynomial $B_i^n(u)$ for $n = 2$, defined as

$$B_i^n(u) \ = \ \frac{n!}{(n-i)!i!}(1-u)^{n-i}u^i. \tag{6}$$

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:

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 3, where $\mathbf{u}_a$ and $\mathbf{u}_b$ are the parameter space tuples of $\mathbf{a}$ and $\mathbf{b}$, 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

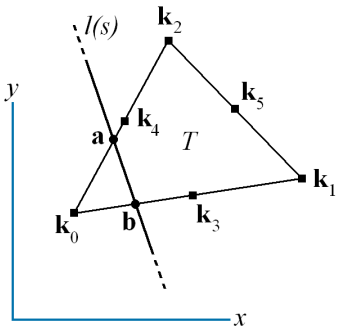

Figure 3: *Intersection of line $l(s)$ with linear-edge quadratic triangle $T(u,v)$ at points **a** and **b**.*

polynomial defined over T (abbreviated as T^c) sampled at locations $\{\mathbf{u}_a, \frac{\mathbf{u}_a+\mathbf{u}_b}{2}, \mathbf{u}_b\}$. 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 $C^c\left(\frac{1}{2}\right) = T^c\left(\frac{\mathbf{u}_a+\mathbf{u}_b}{2}\right)$—is defined as

$$c_1 = 2\,T^c\left(\frac{\mathbf{u}_a+\mathbf{u}_b}{2}\right) - \frac{T^c(\mathbf{u}_a)+T^c(\mathbf{u}_b)}{2}. \qquad (7)$$

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 the polynomial defined over an intersection segment.

4.2. 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 $T(u,v,w)$. 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.

5. 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.

5.1. 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 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, 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

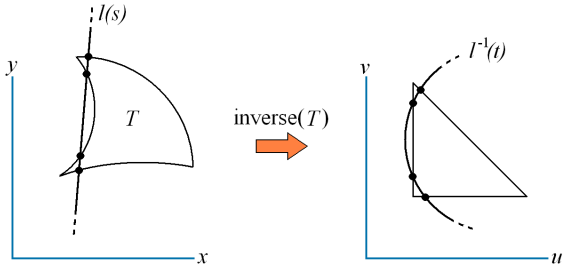

Figure 4: *Transformation of line $l(s)$ in xy-space to $l^{-1}(t)$ in uv-space by the inverse transformation of the curved-quadratic triangle $T(u,v)$.*

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 $\mathbf{p}_m = (x_m, y_m)^{\mathrm{T}}$, $0 \le m \le N-1$.
2. Reorder the points $\mathbf{p}_m$ sequentially based upon the distance from the viewpoint.
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 [BP93]. (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

$$Q(t) = \mathbf{r}_0 t^2 + \mathbf{r}_1 t + \mathbf{r}_2, \qquad (8)$$

and $l(s)$ is in the form

$$l(s) = \mathbf{q}_0 + \mathbf{q}_1 s, \tag{9}$$

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

$$\begin{aligned} 0 \;=\; & (x_0^r y_1^q - y_0^r x_1^q)t^2 + \\ & (x_1^r y_1^q - y_1^r x_1^q)t + \\ & x_2^r y_1^q + x_1^q y_0^q - x_0^q y_1^q - x_1^q y_2^r \end{aligned} \tag{10}$$

and

$$s(t) = \frac{(x_0^r y_1^r - y_0^r x_1^r)t + y_0^r x_0^q + y_2^r x_0^r - y_0^r x_2^r - y_0^q x_0^r}{x_0^r y_1^q - y_0^r x_1^q}. \tag{11}$$

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

Step 2 Labeling 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, $0 \le i \le 2$, distributed uniformly across its domain $t \in [0,1]$, is defined as

$$C(t) = \sum_{i=0}^{2} \mathbf{u}_i B_i^2(t), \tag{12}$$

where B_i^2 is given by Equation (6).

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 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

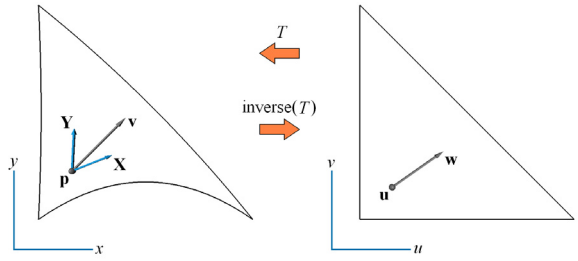

Figure 5: *Inverse transformation of vector* $\boldsymbol{v}$*—located at point* $\boldsymbol{p} = T(\boldsymbol{u})$*—through curved-quadratic mapping* $T(u,v)$*. Vector* $\boldsymbol{v}$ *is transformed from physical space to vector* $\boldsymbol{w}$ *in parameter space by considering the linear combination* $\boldsymbol{v} = \alpha\boldsymbol{X} + \beta\boldsymbol{Y}$*, where* $\boldsymbol{w} = (\alpha, \beta)^{\mathrm{T}}$*.*

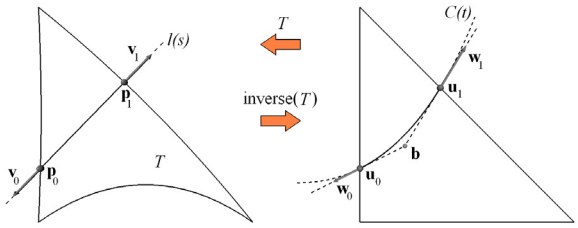

Figure 6: *Quadratic curve* $C(t)$ *approximates the inverse of line* $l(s)$ *based on the curved-quadratic mapping* $T(u,v)$*.*

$\mathbf{u}_{m+1}$, respectively—are intersected 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.

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

$$\begin{aligned} c_0 \;&=\; T^c\left(C(0)\right) = T^c(\mathbf{u}_i), \\ c_1 \;&=\; 2\,T^c\left(C\left(\frac{1}{2}\right)\right) - \frac{T^c(\mathbf{u}_i) + T^c(\mathbf{u}_{i+1})}{2}, \text{ and} \\ c_2 \;&=\; T^c\left(C(1)\right) = T^c(\mathbf{u}_{i+1}), \end{aligned} \tag{13}$$

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.

5.2. 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 $Q_e(u,v)$. A simple closed form solution causes problems (as in the 2D case) since it may not exist, has several solutions (possibly imaginary), and requires extensive computation. An iterative or adaptive method based upon some error requirement could also be used, however, it is simpler to tessellate the face uniformly instead of re-computing an adaptive tessellation for each $l(s)$.

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 5.1. 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 5.1.

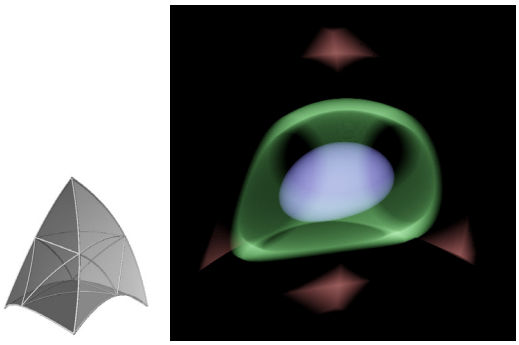

Figure 7: *Ray casting of curved-quadratic tetrahedron $T(u,v,w)$ having non-uniform density—where corner coefficients are zero and edge coefficients are one. Left image shows a rendering of the curved "faces" of $T(u,v,w)$. Right image shows ray casting of $T(u,v,w)$. Isosurfaces are for a small range 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.*

6. Ray Casting Results

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 7. The 487×473 pixel image required 35 seconds to compute. Ray intersections were computed by first tessellating each "face" of the curved-quadratic tetrahedron with 400 linear triangles and then intersecting with these triangles. 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.

A ray casting of 320 curved-quadratic tetrahedra approximating $f(x,y,z) = \frac{3}{4}\left(x^2+y^2+z^2\right), -\frac{1}{2} \leq x,y,z \leq \frac{1}{2}$ is shown in Figure 8. The curved tetrahedra are produced by twisting the mesh a quarter turn from the top to the bottom as indicated in the upper image. The 766×717 pixel image required 50 minutes to compute. Ray intersections were computed by first tessellating each "face" of the curved-quadratic tetrahedron with 400 linear triangles and then intersecting with these triangles. Isosurfaces are for a small range of values near $\{0.2, 0.43, 0.5\}$. The "sphere" corresponds to the isosurface at 0.2. A comparable linear representation of this mesh could be created by tessellating each curved tetrahedron with $8^4 = 4096$ linear tetrahedra. This yields 256 linear tetrahedra representing each "face" (compared to 400). One would then need to ray cast 1310720 unstructured linear tetrahedra.

Back-to-front compositing was used to accumulate several uniform samples along each ray. A pixel intensity $I_n^{i,j}$

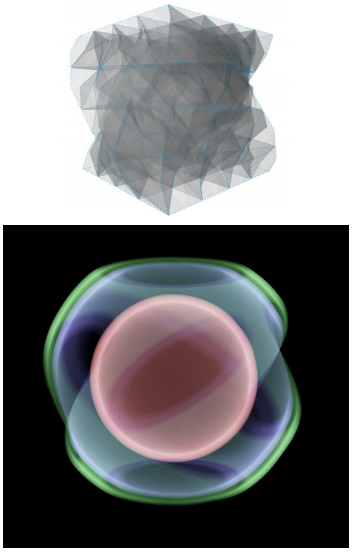

Figure 8: *Ray casting of* 320 *curved-quadratic tetrahedra approximating* $f(x,y,z) = \frac{3}{4}\left(x^2+y^2+z^2\right), -\frac{1}{2} \leq x,y,z \leq \frac{1}{2}$. *The curved tetrahedra are produced by twisting the mesh a quarter turn from the top to the bottom. Isosurfaces are for a small range of values near* $\{0.2, 0.43, 0.5\}$. *The "sphere" corresponds to the isosurface at* 0.2.

computed from the n^{th} sample $c_n \in [0,1]$ is given by

$$I_n^{i,j} = I_{n-1}^{i,j} D(1-c_n), \tag{14}$$

where D is the sampling distance and $(i,j)^{\mathrm{T}}$ is the location of the pixel. A binary space partition using oriented bounding boxes around the triangles was used to optimize ray-triangle intersection tests. All examples were computed on a 2.8GHz Pentium IV graphics workstation with 2GB of main memory.

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 plane 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. 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.)

8. 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 9. 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.

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

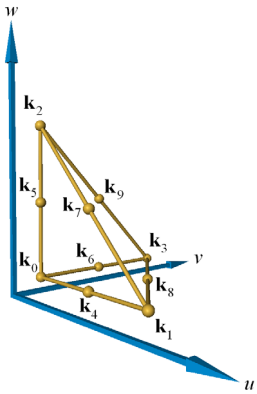

Figure 9: *Indexing used for curved-quadratic tetrahedron.*

its knots to the locations given by

$$\begin{aligned}
\mathbf{v}_0 &= \mathbf{u}_0, \\
\mathbf{v}_1 &= \mathbf{u}_1, \\
\mathbf{v}_2 &= \mathbf{u}_2, \\
\mathbf{v}_3 &= \frac{\mathbf{u}_0+\mathbf{u}_1}{2}, \\
\mathbf{v}_4 &= \frac{\mathbf{u}_0+\mathbf{u}_2}{2}, \text{ and} \\
\mathbf{v}_5 &= \frac{\mathbf{u}_1+\mathbf{u}_2}{2}.
\end{aligned} \tag{15}$$

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

$$\begin{aligned}
c_0 &= T^c(\mathbf{u}_0), \\
c_1 &= T^c(\mathbf{u}_1), \\
c_2 &= T^c(\mathbf{u}_2), \\
c_3 &= 2T^c\left(\frac{\mathbf{u}_0+\mathbf{u}_1}{2}\right) - \frac{T^c(\mathbf{u}_0)+T^c(\mathbf{u}_1)}{2}, \\
c_4 &= 2T^c\left(\frac{\mathbf{u}_0+\mathbf{u}_2}{2}\right) - \frac{T^c(\mathbf{u}_0)+T^c(\mathbf{u}_2)}{2}, \text{ and} \\
c_5 &= 2T^c\left(\frac{\mathbf{u}_1+\mathbf{u}_2}{2}\right) - \frac{T^c(\mathbf{u}_1)+T^c(\mathbf{u}_2)}{2},
\end{aligned} \tag{16}$$

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 relabeling the intersection points $\mathbf{p}_m$ and $\mathbf{u}_m$ with respect to each triangle, then, using Equation (16) finds the coefficients.

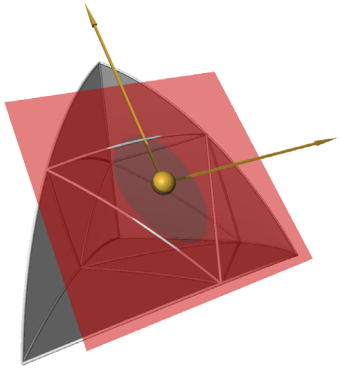

Figure 10: *Intersection of plane and curved-quadratic tetrahedron. Difficulties arise since the plane can intersect the tetrahedron without intersecting its edges. (In this paper, quadratic tetrahedra are often rendered with parametric lines on the faces of the element. These lines help show the curvature of the element. Plane R intersects the parametric lines shown on the faces of tetrahedron T, however, these lines are not as useful as the edges of the tetrahedron.)*

9. 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 10 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 11. The cutting plane is visualized by rendering this uniform rectilinear representation.

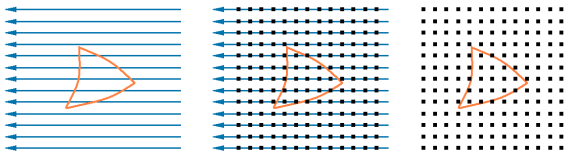

Figure 11: *Discrete sampling of cutting plane. Rays are cast through data and then sampled discretely to construct a uniform rectilinear grid that represents the cutting plane. Left image shows rays—lying on the cutting plane—used to intersect elements. Middle image shows discretely sampled rays. Right image shows resulting uniform rectilinear grid used for visualization.*

10. Cutting Plane Results

Two cutting planes through a single curved-quadratic tetrahedron—where corner coefficients are zero and edge coefficients are one—are shown in Figure 12. Each cutting plane visualization used the ray casting method described in Section 5.2 as applied in Section 9 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 intersection testing. 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.) Each cutting plane visualization required 12 seconds to compute on a 2.8GHz Pentium IV graphics workstation with 2GB of main memory.

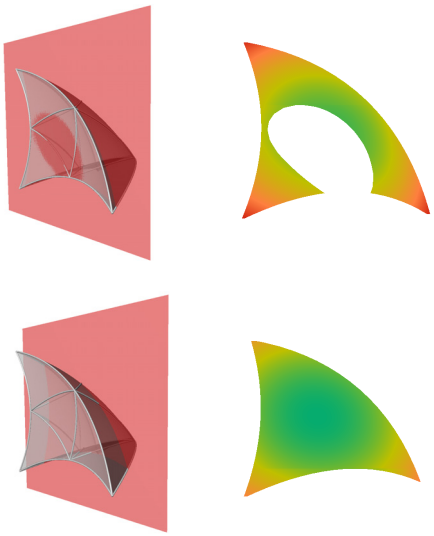

Figure 12: *Two cutting planes intersecting curved-quadratic tetrahedron.*

11. Conclusions

The method presented allows for direct volume rendering of curved-quadratic elements. These ideas could be extended to elements of higher order than quadratic. The "descriptive power" of higher-order elements is sparking interest in researchers and we must have a means to visualize these types of elements. This ray casting method lays the ground work for direct volume rendering of complex higher-order elements and provides a roadway to elements having a higher order than quadratic.

In some cases it may be appropriate to tessellate the faces of a curved tetrahedron with more linear triangles to provide more accurate intersection computation between a ray and an element. Additionally, one may want to use a higher order approximation of the functional over intersection segments. These enhancements will improve close-up views of elements.

Acknowledgements

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the National Institute of Mental Health and the National Science Foundation under contract NIMH 2 P20 MH60975-06A2; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; and the Lawrence Berkeley National Laboratory. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

[BP93] BOEHM W., PRAUTZSCH H.: *Numerical Methods*. A.K. Peters, Ltd., Wellesley, Massachussets, 1993. 4

[CMP89] COOK R., MALKUS D., PLESHA M.: *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, New York, 1989. 1

[Far02] FARIN G.: *Curves and Surfaces for Computer Aided Geometric Design*, fifth edition ed. Academic Press, San Diego, CA, 2002. 2

[Kau91] KAUFMAN A.: *Volume Visualization*. IEEE Computer Society Press, Los Alamitos, CA, 1991. 2

[PPL*99] PARKER S., PARKER M., LIVNAT Y., SLOAN P., HANSEN C., SHIRLEY P.: Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 238–250. 2

[WCG*03] WILEY D., CHILDS H., GREGORSKI B., HAMANN B., JOY K.: Contouring curved quadratic elements. *Data Visualization 2003, Proceedings of VisSym 2003, G.-P. Bonneau, S. Hahmann, and C.D. Hansen, eds.* (2003), 167–176. 2

[WCH*02] WILEY D., CHILDS H., HAMANN B., JOY K., MAX N.: Best quadratic spline approximation for hierarchical visualization. *Data Visualization 2002, Proceedings of VisSym 2002, D. Ebert, P. Brunet, and I. Navazo, eds.* (2002), 133–140. 1

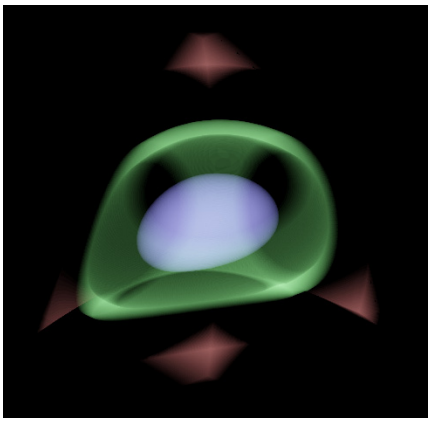

Figure 7: Ray casting of curved-quadratic tetrahedron $T(u,v,w)$ having non-uniform density—where corner coefficients are zero and edge coefficients are one. Top image shows a rendering of the curved "faces" of $T(u,v,w)$. Bottom image shows ray casting of $T(u,v,w)$. Isosurfaces are for a small range 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.

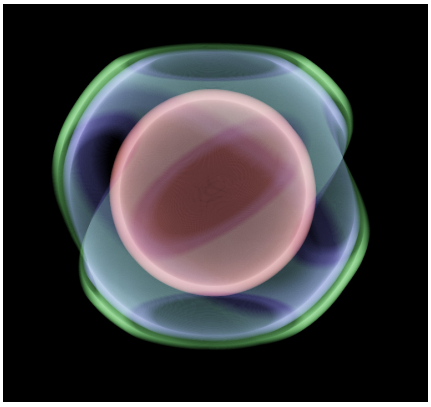

Figure 8: Ray casting of 320 curved-quadratic tetrahedra approximating $f(x,y,z) = \frac{3}{4}\left(x^2+y^2+z^2\right), -\frac{1}{2} \leq x,y,z \leq \frac{1}{2}$. The curved tetrahedra are produced by twisting the mesh a quarter turn from the top to the bottom. Isosurfaces are for a small range of values near $\{0.2, 0.43, 0.5\}$. The "sphere" corresponds to the isosurface at 0.2.