Projected Tetrahedra Revisited: A Barycentric Formulation Applied to Digital Radiograph Reconstruction Using Higher-Order Attenuation Functions

Ofri Sadowsky, *Student Member*, *IEEE*, Jonathan D. Cohen, and Russell H. Taylor, *Fellow*, *IEEE*

Abstract—This paper presents a novel method for volume rendering of unstructured grids. Previously [1], we introduced an algorithm for perspective-correct interpolation of barycentric coordinates and computing polynomial attenuation integrals for a projected tetrahedron using graphics hardware. Here, we enhance the algorithm by providing a simple and efficient method to compute the projected shape (silhouette) and tessellation of a tetrahedron, in perspective and orthographic projection models. Our tessellation algorithm is published here for the first time. Compared with works of other groups on rendering unstructured grids, the main contributions of this work are: 1) A new algorithm for finding the silhouette of a projected tetrahedron. 2) A method for interpolating barycentric coordinates and thickness on the faces of the tetrahedron. 3) Visualizing higher-order attenuation functions using GPU without preintegration. 4) Capability of applying shape deformations to a rendered tetrahedral mesh without significant performance loss. Our visualization model is independent of depth-sorting of the cells. We present imaging and timing results of our implementation, and an application in time-critical "2D-3D" deformable registration of anatomical models. We discuss the impact of using higher-order functions on quality and performance.

Index Terms—Volume rendering, unstructured grids, projected tetrahedra, DRR, higher-order volumetric functions.

1 Introduction

DRR (digitally reconstructed radiograph) generation is a volume rendering technique that simulates the x-ray imaging process by computing integrals of a volumetric attenuation function along the lines of sight. DRRs are commonly used in computer assisted surgery (CAS), with the attenuation function consisting of a voxelized CT study, which is viewed from different directions. An example application of DRRs is intraoperative "2D-3D" registration, i.e., finding the pose of the preoperative CT dataset given a small number of patient radiographs (target images), each with a known camera pose. It works by maximizing the similarity between the target images and DRRs generated dynamically from the CT study. In the surgical setup, a fast registration method is critical. Existing CAS techniques usually visualize voxel grids and not unstructured grids. A specific advantage of using an unstructured grid is that deformations are more easily applied to it than to a voxel set. This enables using statistical models as an augmentation to specific patient scans. However, so far, there has not been a technique for fast enough rendering of a deformable unstructured grid. This paper attempts to provide one. Other uses of DRRs are in physical simulations of x-ray experiments, such as [2].

• *The authors are with the Department of Computer Science, NEB-224, Johns Hopkins University, 3400 N. Charles St., Baltimore, MD 21218. E-mail: {ofri, cohen, rht}@cs.jhu.edu.*

Manuscript received 11 Nov. 2005; revised 12 Jan. 2006; accepted 16 Jan. 2006; published online 10 May 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0159-1105.

This paper follows up on our previous publication [1], developing an algorithm for generating DRRs of a tetrahedral mesh. As in [1], each cell in the mesh contains a representation of the local values of the attenuation function as a higher-order Bernstein polynomial in barycentric coordinates. The DRR visualization model computes the line integrals of the function using a closed formula. This paper extends [1] with a novel method of computing the projected 2D shape of a tetrahedron (triangle or quadrilateral), tessellating the shape into three or four facets, and computing relevant vertex attributes, by exploiting properties of the barycentric representation of spatial coordinates. We propose a simple and elegant alternative to the classic Projected Tetrahedra (PT) method, as presented by Shirley and Tuchman [3], and to its developments by others. Our method is applicable in two conventional linear projection models: perspective and orthographic, and we present the solution for each. We believe that its principles can be expanded to other projection models, and discuss that in Section 7. Note that the new PT algorithm has been developed after the final submission of [1], and is therefore a new contribution published here in detail for the first time.

A simple example of rendering one tetrahedron with a higher-order density function under perspective projection is shown in Fig. 1. Notice that the density is high near the vertices, and low near the center. This cannot be achieved with a constant or linear density function.

2 Related Work

This work was developed in the context of a statistical atlas of bone anatomy, continuing the work of Yao [4]. In his

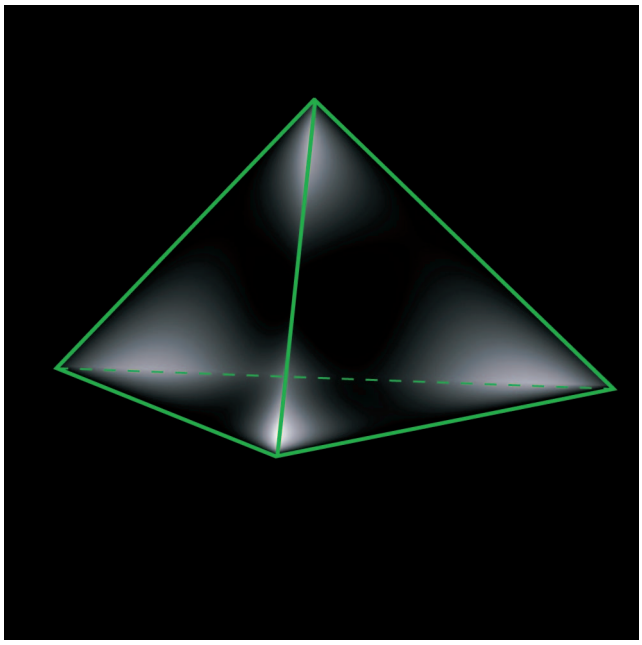


Fig. 1. A single tetrahedron with a third-order density function, rendered in perspective. The locations of the edges are highlighted in green. The density function has high values near the vertices, and low values around the center.

work, Yao represented the shape of the bone as a tetrahedral mesh, and the attenuation function as barycentric Bernstein polynomials for each cell of the mesh. Tetrahedral meshes maintain their topology when deformations are applied, and the barycentric polynomials provide a local density distribution function which is independent of the cell deformation. We follow this representation in the current work.

One goal of the statistical atlas was to provide a model for deformable 2D-3D registration between x-ray images of a patient and dynamic DRRs of the statistical atlas. Yao implemented a DRR generator using a discrete approximation of the integrals in a ray-tracing-like manner. However, the use of polynomials provides a closed formula for the line integrals in a tetrahedron. Our work uses this formula for a more efficient and accurate computation of the DRR images.

The Projected Tetrahedra (PT) algorithm, presented by Shirley and Tuchman [3], is the basis to many works on rendering unstructured grids. In their paper, a tetrahedral mesh is created from a rectilinear voxel grid by subdividing the voxels. The volumetric function is assumed to interpolate linearly between the mesh vertices. The rendering is done by depth-sorting the tetrahedra and breaking each one into three or four triangular facets (tessellation), which are sent to the rendering pipeline and blended in the frame buffer. The integral is computed at the vertices, and interpolated linearly from the thick point of the projected tetrahedron to its boundary. Their method is not geometrically correct, as noted in [5], and we address that below.

Wylie et al. [6] implemented the PT tessellation algorithm using a GPU vertex shader. They report that their performance results were limited by the capacities of the graphics hardware at the time. Their tessellation algorithm assumes orthographic projection, and they still use linear interpolation of the thickness property, again assuming orthographic projection. Further discussion of the tessellation step in PT and its implementations is in Section 4.

A number of volume rendering methods use precomputed line integrals to accelerate the rendering. So-called "light field" methods [7], [8] are one set of examples. In the particular context of DRR-based registration, LaRose [9] presented a rendering method that uses a four-dimensional array, indexed by pairs of planar coordinates, as a lookup table for fetching integral values. The method renders a full voxelized dataset by using the intersections of lines of sight with the volume boundary as the lookup table indexes, and was applied in *rigid* 2D-3D registration. Röttger et al. [10] use precomputed integrals to render a tetrahedral mesh. The volumetric function is assumed to be interpolated linearly between its values at the mesh vertices (in that sense, it is *piecewise linear*), and includes color, attenuation, and emission. Under an orthographic projection model, they interpolate the intersection points of each tetrahedron with the lines of sight, and use a resulting triad of coordinates as an index to a 3D texture, i.e., a lookup table, to fetch the precomputed integral.

Perspective projection of tetrahedra was presented by Weiler et al. [11]. They compute the intersection endpoints of a ray and a cell by solving line-plane-intersection equations, a method also adopted by Moreland and Angel [12]. The main drawback of this method was its complexity, making implementation on a GPU fragment unit difficult. Kraus et al. [5] followed by presenting a method to interpolate the thickness and intersection points under a perspective projection model, still using preintegration. Their per-vertex data includes two sets of coordinates, for the front and back sides of the facet, which are interpolated on the GPU.

Methods that render an unstructured grid with volumetric functions that combine emission and attenuation require depth sorting of the cells. Sorting algorithms have been presented, such as [13], [14], [15]. A fast method that combines visibility sorting of polygons and fragments of an unstructured grid, and rendering a preintegrated function, was recently presented by Callahan et al. [16]. The volumetric function in our work includes attenuation only and, therefore, is simpler than some of the works cited above, and does not require sorting. However, our volumetric function is *piecewise polynomial*, which generalizes the linear model to a higher degree, allows a compact approximated representation of complicated structures, and facilitates enforcement of continuity constraints across cell boundaries. Note that there is a tradeoff between polynomial degree and mesh resolution, and that our actual models do not include continuity constraints. However, these issues are not related to the rendering algorithm we present here. We did not find previous works on computing line integrals of high-order functions on the GPU fragment unit. To obtain the higher-order functions, we generate a mesh such that each tetrahedron covers multiple voxels of an input volume (see [17]), and find a polynomial function that approximates the voxel values inside.

Barycentric coordinates are a key component in both our volumetric model and our algorithm. None of the works cited above use barycentric coordinates in their models. Interpolation of the barycentric coordinates of the fragments was a central part in [1]. However, the method for finding the barycentric coordinates at the vertices in [1] was highly inefficient. Here, we present a simple and efficient solution to this problem. While our volumetric function is monochromatic, the use of barycentric coordinates to represent a weighted combination of the vertices would facilitate interpolation of color attributes as well.

Compared to other works on rendering unstructured grids, cited above, and to our previous paper, the main contributions in this paper are:

1. A simple and efficient algorithm for finding the shape, the vertices and the vertex attributes of a projected tetrahedron, applied in different projection models.
2. A method to interpolate the intersection endpoints of a ray and a tetrahedron, including barycentric coordinates and thickness, applied in different projection models.
3. Visualizing line integrals of higher-order attenuation functions.
4. Visualizing *deformable* tetrahedral meshes without significant performance loss in the deformation.

In addition, due to the imaging model we use, our method is independent from preintegration and depth sorting.

3 Algorithm Outline

We render a tetrahedral mesh with polynomial density functions for the cells by rendering each cell and accumulating its output fragment values to the rendering target. The process does not depend on visibility order, as our volumetric model represents attenuation only. However, components and principles in our algorithm can be used in volumetric models that combine both attenuation and emission. For example, the tessellation algorithm is universal, and barycentric coordinates can potentially parametrize other volumetric functions and their projections.

Rendering a single tetrahedron includes the following steps:

1. Find the projected shape of the tetrahedron on the image plane. The shape may be a triangle or a quadrilateral.
2. Determine the "central" point of the projected shape, which we call the *thick point*, and the boundary vertices. Each vertex has attributes such as position, barycentric coordinates, tetrahedron index, etc.
3. Generate three or four triangular facets, each including the thick point and two boundary vertices. Send the facets to the rendering pipeline.
4. Transform and rasterize each triangle on the GPU, interpolating the vertex attributes for each generated fragment.
5. Compute the line integral at each fragment in a *fragment program*. This involves reading a list of coefficients for the basis functions of the integral from a texture object, then computing the integral using a closed formula.
6. Accumulate the fragment in a high-dynamic-range pbuffer to form the final image.

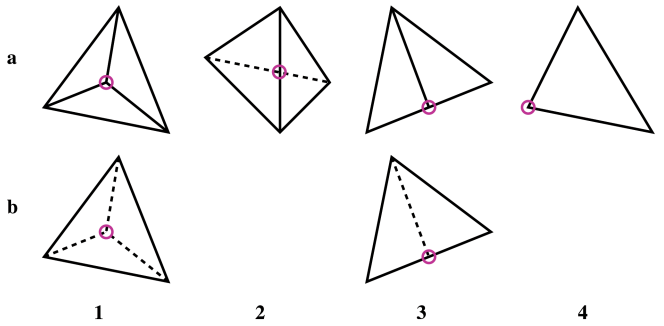


Fig. 2. Classification of the shapes of a projected tetrahedron by Shirley and Tuchman [3]. The *thick point* is highlighted with a magenta circle.

Our previous paper, [1], focused mainly on correct interpolation of fragment attributes, and computing the integral formula, which comprise steps 3 to 6 above. These are summarized in Section 5. This paper focuses on the details of a new tessellation algorithm, which is steps 1 and 2, presented next in Section 4.

4 Projected Tetrahedron

4.1 Prior Arts

Shirley and Tuchman's version of the PT method [3] classified the silhouette of a projected tetrahedron into six cases, illustrated in Fig. 2. To identify the case, they used the directions of the four face normals with respect to the viewer. The projection (dot product) of the normal on the view vector may be positive, negative, or zero, and by counting the signs of the four projections the case is determined. The projected shape is split into one to four triangular facets, pivoted around the thick point (highlighted with a magenta circle in the figure). To find the thick point, they interpolate the coordinates of the point opposite the thick vertex in a triangle case, and compute line intersections in the plane for a quadrilateral shape. Their solution assumes that the near and far endpoints of the thick line have the same x and y coordinates in eye space. This is correct only under orthographic projection. The facets are rendered by linearly interpolating the line integrals between the thick point and the boundary, which is correct only with respect to a limited set of volumetric functions, e.g., additive piecewise constant functions, under orthographic projection.

Wylie et al. [6] presented a method that computes the shape and the tessellation of a projected tetrahedron in a GPU *vertex program*. To overcome the limitations of the GPU architecture, they use quadrilateral topology (Fig. 2, case 2) as a superclass for all the other cases, which will be rendered correctly with some additional degenerate primitives. Because vertices on the GPU are processed independently, rendering the quadrilateral as a triangle fan requires six invocations of the tessellation code in the vertex processing unit, and some redundant work is performed.

Each invocation determines the shape of the silhouette, the thick point, and the final attributes of all the output vertices, and then selects the attribute tuple of the appropriate vertex to output. They facilitate this process on the GPU by identifying 14 permuted configurations of a canonical triangle fan and performing the permutation using predefined permutation matrices.

For the purpose of comparing Wylie's algorithm with our CPU approach, we can ignore most of the GPU-specific details, such as the need to invoke the vertex program six times, and just consider the tessellation algorithm itself. Like Shirley and Tuchman, the algorithm first projects the four vertices to clip-space coordinates, then performs four boolean geometric tests and reorders the vertices into a *rendering order*. Line intersections are computed in the XY plane without perspective division. The thickness of the thick line is determined as the difference in Z coordinates of the front and back endpoints, and interpolated linearly from it to the boundary vertices. Again, both the computation of the thick vertex and the interpolation of the thickness are correct only under orthographic projection.

Weiler et al. [11] presented a different strategy. Their method is based on rendering the four faces of the tetrahedron, having the back-side faces culled. It does not require tessellation, and is view independent. Instead, they show how the thickness can be computed in a fragment program which is given the plane parameters of three faces. At the time, hardware that could perform these computations for perspective projection was unavailable and, therefore, their implementation is for orthographic projection, which enables them to take advantage of the linearity property of the thickness. Following that, Kraus et al. [5] presented the first method we could find for perspective-correct interpolation of the thickness and intersection endpoints, using front- and back-side coordinates per vertex. They used a preintegrated volumetric function, parametrized by front and back "scalar values" and the intersection length. As we note above, our volumetric function is piecewise polynomial in barycentric coordinates, and therefore poses other problems.

Generalization of Shirley and Tuchman's PT method, or Wylie's tessellation algorithm, to perspective projection is nontrivial. In [1], we showed two ways for perspective-correct thickness interpolation (see also Section 5). Our attempt in [1] to implement line-intersection tests similar to Wylie's for the perspective case proved much more difficult. It required perspective division before finding the projection of the thick point, and then back projection and computation of line intersections in space to find the true coordinates and thickness at the thick vertex.

The solution proposed here to the Projected Tetrahedron problem is simpler than the works above. In a way, it generalizes observations in both [3] and [6]. We solve the problem under perspective projection, and with a slight variation under orthographic projection, using similar principles in both. The inclusion of barycentric coordinates in the vertex attributes could, potentially, increase the algorithm's complexity in comparison to the "standard" geometrical models. Surprisingly, the derivation of the solution in barycentric coordinates contributes to its efficiency.

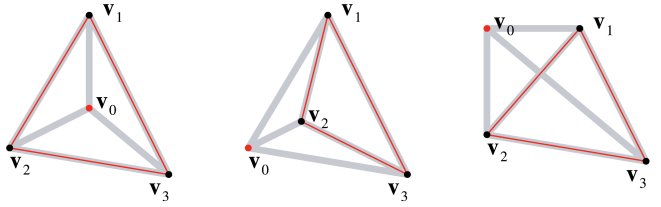


Fig. 3. A few configurations of the vertex $\mathbf{v}_0$ and the tetrahedral face $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$, highlighted in red, projected on the image plane. The edges of the tetrahedron are shown as thick gray lines.

To simplify the presentation, we first explain the algorithm assuming all the vertices are given in eye-space, i.e., have been transformed by the model-view transformation. In Section 4.4, we show how the same algorithm is applied when the model-view transform is given.

4.2 PT in Perspective Projection

Before giving the full details of the algorithm, we would like to provide an intuition based on a familiar case. Consider the four vertices of the tetrahedron, $\mathbf{T} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$, projected to the image plane. To determine the projected configuration, we wish to know where $\mathbf{v}_0$ is with respect to the face $\mathbf{F}_0$ formed by the other three vertices. Case examples are shown in Fig. 3.

We can characterize each case uniquely using the *barycentric coordinates*[1] of the projected $\mathbf{v}_0$ with respect to the projected $\mathbf{F}_0$ as follows: Let us write the homogeneous coordinates of the vertices in the plane: $\mathbf{p}_i = P(\mathbf{v}_i) = (x_i, y_i, 1)^T$, where $P(\cdot)$ is the normalizing projection transformation, i.e., includes perspective division. We form the face matrix $M_{\mathbf{F}_0} = [\mathbf{p}_1 \ \ \mathbf{p}_2 \ \ \mathbf{p}_3]$. Every point $\mathbf{p}$ in the plane can be represented as a barycentric combination $\mathbf{p} = q_1\mathbf{p_1} + q_2\mathbf{p}_2 + q_3\mathbf{p}_3 = M_{\mathbf{F}_0}\mathbf{q}$, $\mathbf{q} = (q_1, q_2, q_3)^T$, with $q_1 + q_2 + q_3 = 1$. The barycentric coordinates of $\mathbf{p}_0$ with respect to the face are therefore: $\mathbf{q}_0 = M_{\mathbf{F}_0}^{-1}\mathbf{p}_0$. We now observe that the image plane is divided by the triangle $\mathbf{F}_0$ into nineteen regions (seven nondegenerate 2D regions, and 12 1D or 0D regions), according to the signs of the q_j's (some of the q_j's are zero in the degenerate regions), as illustrated in Fig. 4.

To extend this observation to the 3D space in a perspective projection model, we compute the barycentric coordinates of the *eye position* with respect to a tetrahedron. Having the vertices $(\mathbf{v}_0, \ldots, \mathbf{v}_3)$ given in homogeneous coordinates, $\mathbf{v}_i = (x_i, y_i, z_i, 1)^T$, we form the vertex matrix $M_{\mathbf{T}}$, which transforms barycentric coordinates to homogeneous coordinates; its inverse $M_{\mathbf{T}}^{-1}$ transforms homogeneous coordinates to barycentric coordinates:

$$M_{\mathbf{T}} = [\,\mathbf{v}_0 \quad \mathbf{v}_1 \quad \mathbf{v}_2 \quad \mathbf{v}_3\,]. \tag{1}$$

Given the eye position (or x-ray *source*) as $\mathbf{s} = (0, 0, 0, 1)^T$, its barycentric coordinates are $\mathbf{T} = M_{\mathbf{T}}^{-1}\mathbf{s} = (t_0, t_1, t_2, t_3)^T$. The tetrahedron $\mathbf{T}$ divides the space into 65 regions, 15 of which are nondegenerate 3D regions, and the others are

1. In this paper, barycentric coordinates are not sign-restricted. The only constraint on the barycentric coordinates of a *point* is that their sum is 1. The barycentric coordinates of a *direction vector* sum to 0.

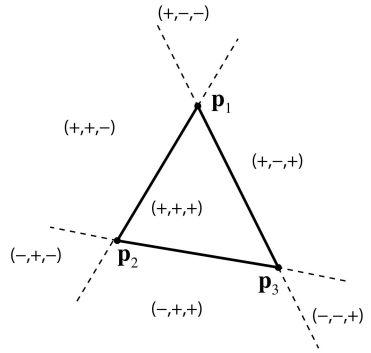


Fig. 4. Signs of barycentric coordinates with respect to a triangle in the plane, shown for seven nondegenerate 2D regions. On the vertices and (open) edges of the triangle, and on the rays extending along edge directions outwards from the vertices, some components are zero.

along the lower-dimension elements of $\mathbf{T}$.[2] Ignoring the degenerate cases (assuming they are automatically culled in the rendering pipeline), we turn to examine the signs of the t_j's and observe the following.

- All t_j are nonnegative *if and only if* $\mathbf{s}$ is in the interior or on the boundary of $\mathbf{T}$. In this case, we do not render $\mathbf{T}$.
- One t_{j_0} is strictly positive, while the others are nonpositive, *if and only if* $\mathbf{s}$ is in front of $\mathbf{v}_{j_0}$, which is on the thick line (cases 1a, 3a, and 4 in Fig. 2).
- One t_{j_0} is strictly negative, while the others are nonnegative, *if and only if* $\mathbf{s}$ is in front of the face $\mathbf{F}_{j_0}$, and $\mathbf{v}_{j_0}$ is on the thick line (cases 1b and 3b in Fig. 2).
- t_{j_0} and t_{j_2} have one sign, while t_{j_1} and t_{j_3} have the opposite sign, *if and only if* the silhouette is a quadrilateral, $\mathbf{s}$ is in front of both edges $(\mathbf{v}_{j_0}, \mathbf{v}_{j_2})$ and $(\mathbf{v}_{j_1}, \mathbf{v}_{j_3})$, and the thick line intersects both (case 2 in Fig. 2).

Notice that the 15 nondegenerate regions correspond to Wylie's 14 configurations, plus the interior-point case, which is not renderable. To determine the projected case of the silhouette, it is enough to compute $\mathbf{T}$, then count the positive and negative components in it. This takes about the same number of operations as Shirley and Tuchman's checking the relative directions of face normals and view direction, and is close in meaning. However, we draw more information from this result than from testing the normals, as we can find directly the near and far endpoints of the thick line without computing line intersections.

First, let us reorder the vertices into *rendering order*, so that in a triangle silhouette, $\mathbf{v}_0$ is on the thick line (near or far), and in a quadrilateral silhouette, $\mathbf{v}_0$ and $\mathbf{v}_2$ are at opposite corners, with the edge between them intersecting the (near or far) thick line (see Fig. 5).

Note at this stage that we do not care if we pivot the triangle fan about the near or the far thick vertex, or whether the facets are given in clockwise or counterclockwise winding order, for reasons that will be given later. Therefore, changing from the original winding order of the vertices to the rendering order requires swapping at most one pair of vertices: $\mathbf{v}_0$ and $\mathbf{v}_{j_0}$ in the triangle case, and

2. For a d-simplex in $\mathbf{R}^d$, the total number of regions is given by $r = \sum_{k=1}^{d+1} \binom{d+1}{k}(2^k - 1) = 3^{d+1} - 2^{d+1}$, out of which $2^{d+1} - 1$ are nondegenerate.

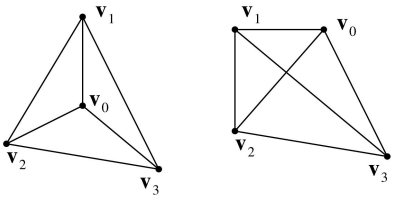


Fig. 5. Rendering order of vertices in (left) triangle and (right) quadrilateral silhouettes of a projected tetrahedron.

$\mathbf{v}_0$ with $\mathbf{v}_{j_0}$, where $j_0 \in \{1, 3\}$, $\operatorname{sign}(t_0) \neq \operatorname{sign}(t_2)$ and $\operatorname{sign}(t_2) = \operatorname{sign}(t_{j_0})$ in a quadrilateral case. We keep track of j_0 for later use. We reorder the vertices (cf. [6]) to simplify the following steps, and assume from here on that the vertices are in rendering order.

The line integral requires knowing the near and the far intersection endpoints of the line of sight and the tetrahedron. In our case, we only need the eye-space coordinates of one endpoint, and the barycentric coordinates of both (see Section 5). We find these attributes using the following equations.

- In a triangle case, we choose $\mathbf{v}_0$ as one of the endpoints of the thick line, with barycentric coordinates $\mathbf{u}_0 = (1, 0, 0, 0)^T$. The other endpoint is $\mathbf{b} = \alpha \mathbf{v}_0$ (in *Cartesian*, nonhomogeneous coordinates), and lies on the face $\mathbf{F}_0$, so its barycentric coordinates are $\mathbf{u}_b = (0, u_{1b}, u_{2b}, u_{3b})^T$. To find them, we observe that

$$\begin{aligned} \mathbf{b} &= (1-\alpha)\mathbf{s} + \alpha \mathbf{v}_0 \\ \mathbf{u}_b &= (1-\alpha)\mathbf{t} + \alpha \mathbf{u}_0 \\ 0 &= (1-\alpha)t_0 + \alpha \\ \alpha &= \frac{-t_0}{1-t_0}. \end{aligned} \tag{2}$$

- In a quadrilateral case, one thick endpoint, $\mathbf{f}$, is at the intersection of the edge $(\mathbf{v}_0, \mathbf{v}_2)$ and the plane going through $\mathbf{s}, \mathbf{v}_1, \mathbf{v}_3$. This is encoded in the following equation, with the vertices expressed in *Cartesian* coordinates.

$$\alpha_0 \mathbf{v}_0 + (1-\alpha_0)\mathbf{v}_2 = \alpha_1 \mathbf{v}_1 + \alpha_3 \mathbf{v}_3. \tag{3}$$

The solution, given here without its lengthy derivation, is

$$(\alpha_0, \alpha_1, \alpha_3) = (t_0 + t_2)^{-1}(t_0, -t_1, -t_3) \tag{4}$$

$$\mathbf{f} = \frac{t_0}{t_0 + t_2}\mathbf{v}_0 + \frac{t_2}{t_0 + t_2}\mathbf{v}_2 \tag{5}$$

$$= \frac{-t_1}{t_0 + t_2}\mathbf{v}_1 + \frac{-t_3}{t_0 + t_2}\mathbf{v}_3. \tag{6}$$

The other thick endpoint, $\mathbf{b}$, is at the intersection of the edge $(\mathbf{v}_1, \mathbf{v}_3)$ and the plane through $\mathbf{s}, \mathbf{v}_0, \mathbf{v}_2$. A similar solution can be derived for it:

$$\mathbf{b} = \frac{t_1}{t_1 + t_3}\mathbf{v}_1 + \frac{t_3}{t_1 + t_3}\mathbf{v}_3 \tag{7}$$

$$= -\frac{t_0 + t_2}{t_1 + t_3}\mathbf{f}. \tag{8}$$

Note that t_0, t_2 have equal signs, and t_1, t_3 have an opposite sign. Therefore, neither the numerator nor the denominator are zero. Equation (8) shows that $\mathbf{f}$ and $\mathbf{b}$ are scaled versions of one another, and are projected to the same image point. They are also convex combinations of pairs of opposite vertices, and this proves that they are the thick endpoints.

Hence, we have obtained a parametrization of the near and far endpoints of the thick line using the components of $\mathbf{t}$ for both the triangle and the quadrilateral cases. Note that $\mathbf{t}$ represents the barycentric coordinates of the x-ray source $\mathbf{s}$ with respect to the tetrahedron $\mathbf{T}$, and is independent of the view transform. That is, it can be computed without transforming the mesh vertices, as we show in Section 4.4. Knowing the barycentric coordinates of the thick line endpoints and the boundary vertices in *rendering order*, we end by swapping the index-0 component and the index-j_0 component in the "barycentric coordinates" attributes of each vertex, to restore them to the *original winding order*. We will return to them in Section 5.

4.3 PT in Orthographic Projection

A closer look at the behavior of the barycentric coordinates at the thick line endpoints reveals the following fact. Suppose the silhouette is a triangle, and $\mathbf{v}_0$ is a thick endpoint. Its barycentric coordinates are $\mathbf{u}_0 = (1,0,0,0)^T$. The other thick endpoint, $\mathbf{b}$, is on the opposite face, $\mathbf{F}_0$, with barycentric coordinates $\mathbf{u}_b = (0, u_{1b}, u_{2b}, u_{3b})^T$. The difference vector $\mathbf{b} - \mathbf{v}_0$ is therefore expressed in barycentric space as

$$\mathbf{d}' = \mathbf{u}_b - \mathbf{u}_0 = (-1, u_{1b}, u_{2b}, u_{3b})^T.$$

The same difference vector is aligned with the line of sight for $\mathbf{v}_0$ under any projection model. In perspective projection, it is expressed as: $M_{\mathbf{T}}\mathbf{d}' = \alpha'(x_0, y_0, z_0, 0)^T$. In orthographic projection, the line of sight is always parallel to the Z axis. That is,

$$M_{\mathbf{T}}\mathbf{d}' = \alpha'(0,0,z_0,0)^T = \alpha(0,0,1,0)^T$$

(with $\alpha = z_0\alpha'$), or

$$\begin{aligned} \mathbf{d} &= M_{\mathbf{T}}^{-1}(0,0,1,0)^T \\ \mathbf{d}' &= \alpha\mathbf{d}. \end{aligned} \tag{9}$$

Let us compute $\mathbf{d}$ and look at the signs of its components. Observations similar to those made in the perspective model can be made here. If one sign is strictly positive (or negative) and the others are nonpositive (nonnegative) then the corresponding vertex is a thick endpoint and the silhouette it a triangle. If two signs are positive and two are negative then the silhouette is a quadrilateral, with vertices of equal signs projected at opposite corners. Given that the barycentric coordinates of a *point* $\mathbf{p}$ represent the weights applied to the four vertices so that their linear combination is equal to $\mathbf{p}$, we observe that the barycentric coordinates of a *direction vector* represent the change in these weights as we move along the direction. Projecting the Z direction through $M_{\mathbf{T}}^{-1}$, a positive component means motion through the tetrahedron interior toward the corresponding vertex; a negative component means motion in the interior toward the corresponding face. This provides another explanation to our sign analysis in the orthographic case, as well as in the perspective case.

Having found the configuration of the vertices in the image plane, the next steps are quite similar to those in perspective projection. When the vertices are in rendering order, we find the thick endpoints as follows:

- In a triangle case, we already know $\mathbf{u}_0$, and $\mathbf{d} = (d_0, d_1, d_2, d_3)^T$. Given that $\mathbf{b} = \mathbf{v}_0 + \alpha M_{\mathbf{T}}\mathbf{d}$, and the index-0 component of $\mathbf{u}_b$ is 0, it can be calculated that $\alpha = -{d_0}^{-1}$ and, hence,

$$\mathbf{u}_b = {d_0}^{-1}(0, -d_1, -d_2, -d_3)^T.$$

- In a quadrilateral case, one set of barycentric coordinates is $\mathbf{u}_f = (u_{0f}, 0, u_{2f}, 0)^T$ and the other is $\mathbf{u}_b = (0, u_{1b}, 0, u_{3b})^T$. Given that $\mathbf{d} = \mathbf{u}_b - \mathbf{u}_f$ and the barycentric constraints, it can be calculated that

$$\begin{aligned} \mathbf{u}_f &= (d_0 + d_2)^{-1}(d_0, 0, d_2, 0)^T \\ \mathbf{u}_b &= (d_1 + d_3)^{-1}(0, d_1, 0, d_3)^T. \end{aligned} \tag{10}$$

The rest of the coordinates follow from these results.

4.4 View Transform, Morphing, and Implementation

An essential goal of our project is to be able to visualize the tetrahedral model under different view transformations, and subject to shape deformations or morphing. Obviously, the tetrahedron matrix $M_{\mathbf{T}}$ changes under these transformations. However, as long as the view transform is linear, which is normally the case in computer graphics, and no morphing is applied, $M_{\mathbf{T}}^{-1}$ does not need to be recomputed, and can be cached instead.

Given a view transformation matrix F, if $M_{\mathbf{T}}$ is given in model coordinates, the perspective model computation in eyespace becomes

$$\mathbf{t} = (FM_{\mathbf{T}})^{-1}\mathbf{s} = M_{\mathbf{T}}^{-1}(F^{-1}\mathbf{s}).$$

Remember that $\mathbf{s} = (0,0,0,1)^T$. If $F^{-1} = [\mathbf{g}_0\ \mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3]$, then $F^{-1}\mathbf{s} = \mathbf{g}_3$. Therefore, under linear view transformations, we can cache $M_{\mathbf{T}}^{-1}$ for each tetrahedron, then obtain $\mathbf{g}_3$ from the view matrix inverse and find $\mathbf{T}$ for each tetrahedron. In orthographic model, similar observations show that $\mathbf{d} = M_{\mathbf{T}}^{-1}\mathbf{g}_2$.

We implemented the new PT algorithm in a C++ routine. Our new implementation can be summarized as follows:

- Project the view direction vector through $M_{\mathbf{T}}^{-1}$.
- Compute the thick vertex and the boundary vertices, and their attributes.
- Render a triangle fan with three or four facets, by sending the thick vertex and the boundary vertices. The first boundary vertex is repeated at the end of the sequence, so five or six vertices are sent to the GPU.

Compared with our former PT method that was used in [1], the new algorithm is at least nine times faster (see Section 6). The new algorithm does not require projecting the vertices to the image plane, which we did in [1] in order to compute perspective-correct line intersections. We take advantage of this, and our new implementation uses the

GPU vertex program to transform the tetrahedron vertices to eye-space and to clip-space (both results are required). The algorithm itself should be easy to encode as a GPU vertex program, using, for example, Wylie's output-selection technique [6], though we have not done that yet.

When morphing is applied to the tetrahedral mesh, the matrices $\{M_{\mathbf{T}}^{-1}\}$ need to be recomputed. We use our own C++ routine for 4×4 Gauss-Jordan elimination to find the inverse matrices. Potentially, if the mesh vertices are stored in GPU memory, one may consider using GPU programs for inverting matrices. A CPU code may also be optimized in multiple ways, such as using MMX operations. Again, we have not tried to provide the optimal matrix inversion method. More discussion on shape morphing follows in Section 7.

5 Attribute Interpolation and Integration

This section summarizes the methods we use to interpolate fragment attributes and compute line integrals. It is based on the methods we presented in [1], and adds an extension for an orthographic projection model.

5.1 Transfer Function

Our imaging model is based on the attenuation formulas known as Beer's law. In simplified terms, the x-ray flux I_{out} passing through a unit-thick material slab is equal to $\alpha \cdot I_{in}$, with $0 \leq \alpha \leq 1$ being a material property. Along a parametric curve $\mathbf{p}(t)$, with varying material properties, the overall attenuation of energy is henceforth given by the formula

$$\frac{I_{out}}{I_{in}} = e^{-\int_0^1 ||\nabla \mathbf{p}(t)|| \mu(\mathbf{p}(t)) dt}, \tag{11}$$

where $\mu(\mathbf{p}) = -\ln \alpha(\mathbf{p})$ is called the *linear attenuation coefficient*. We will frequently use the term "density" for this number.

In our mesh model, cell density distribution is encoded as a barycentric Bernstein polynomial. The structure of the polynomial is

$$f^j(\mathbf{u}) = \sum_{|\mathbf{k}|=d} \beta_{\mathbf{k}}^j B_{\mathbf{k}}^d(\mathbf{u}) = \sum_{|\mathbf{k}|=d} \beta_{\mathbf{k}}^j \binom{d}{\mathbf{k}} u_0^{k_0} u_1^{k_1} u_2^{k_2} u_3^{k_3}. \tag{12}$$

Here, j is the index of the tetrahedral cell $\mathbf{T}_j$; $\mathbf{u} = (u_0, u_1, u_2, u_3)$ are barycentric coordinates of a point in the cell; $B_{\mathbf{k}}^d$ are the basis functions, with $\mathbf{k} = (k_0, k_1, k_2, k_3)$ being *power index* of the basis function, and $d = |\mathbf{k}| = k_0 + k_1 + k_2 + k_3$ being the degree; $\binom{d}{\mathbf{k}} = \frac{d!}{k_0! k_1! k_2! k_3!}$ is a multinomial factor, resulting from the development of the Bernstein basis; and $\beta_{\mathbf{k}}^j$ is a free coefficient. The collection of basis functions is the same for all cells, and so each cell only needs to store the free coefficients of its local density.

The DRR image is computed by integrating the density function along the lines of sight. The complete imaging model uses the exponential of the line integral, but we will regard the exponential taking as a postprocessing step, which can be applied independently of our algorithm before displaying or saving the image. We focus on computing the line integrals for each rendered tetrahedron, and summing them together for the mesh.

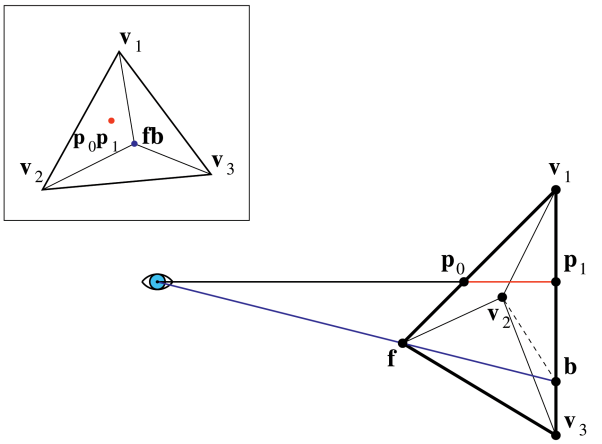


Fig. 6. Points in a projected tetrahedron. The top frame shows the projected view, and the bottom shows a "side view" of the same tetrahedron. $\mathbf{f}$ is the near thick vertex; $\mathbf{b}$ is the far thick vertex; $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ are boundary vertices. The blue line is the thick line of sight, viewed as the blue dot in the top frame. The triangle $(\mathbf{f}, \mathbf{v}_1, \mathbf{v}_2)$ is the front side of a facet; the back side is $(\mathbf{b}, \mathbf{v}_1, \mathbf{v}_2)$. $\mathbf{p}_0$ and $\mathbf{p}_1$ are intersection endpoints of the facet and some line of sight, projected on the red dot in the top frame. The intersection length is highlighted in red.

We compute the integral between the two endpoints, $\mathbf{p}_0$ and $\mathbf{p}_1$, of the intersection between the line of sight and a tetrahedron $\mathbf{T}$ (see Fig. 6). The integral formula for a single basis function is

$$\int_{\mathbf{p}_0}^{\mathbf{p}_1} B_{\mathbf{k}}^d(\mathbf{u}) d\mathbf{u} = \frac{\|\mathbf{p}_1 - \mathbf{p}_0\|}{d+1} \sum_{\mathbf{l} \sqsubseteq \mathbf{k}} B_{\mathbf{l}}^{|\mathbf{l}|}(\mathbf{u}_0) B_{\mathbf{k}-\mathbf{l}}^{|\mathbf{k}-\mathbf{l}|}(\mathbf{u}_1). \tag{13}$$

$\mathbf{l}$ is a power index of degree less than or equal to d, and the notation $\mathbf{l} \sqsubseteq \mathbf{k}$ denotes that $l_i \leq k_i$ for all i. $\mathbf{u}_0$ and $\mathbf{u}_1$ are the barycentric coordinates of $\mathbf{p}_0$ and $\mathbf{p}_1$. The integral includes the intersection length $\|\mathbf{p}_1 - \mathbf{p}_0\|$, the degree d, and the summation of *inner term products* of the form $B_{\mathbf{l}}^{|\mathbf{l}|}(\mathbf{u}_0) B_{\mathbf{k}-\mathbf{l}}^{|\mathbf{k}-\mathbf{l}|}(\mathbf{u}_1)$. Note that the integral does not depend on the order of intersections, i.e., which endpoint is closer to the viewer. This explains why we do not care which thick endpoint is near and which is far.

The final integral of the density function is the sum of the integrals of the basis functions multiplied by the coefficients:

$$\int_{\mathbf{p}_0}^{\mathbf{p}_1} f^j(\mathbf{u}) d\mathbf{u} = \sum_{|\mathbf{k}|=d} \beta_{\mathbf{k}}^j \int_{\mathbf{p}_0}^{\mathbf{p}_1} B_{\mathbf{k}}^d(\mathbf{u}) d\mathbf{u}. \tag{14}$$

We have developed an algorithm for interpolating the relevant attributes from the vertices of the rendered facets to the rasterized fragments. Details of the algorithm and of the Cg fragment program that computes the integrals are given next.

5.2 Interpolating Fragment Attributes

In this part, we show how to interpolate the barycentric coordinates of the intersection endpoints and the intersection length as fragment attributes which are used in the integration formula. Recall from Section 4 that we render three or four facets as a triangle fan about the thick vertex, and that we compute the necessary attributes at the vertices. Let us call the polygon defined by the vertices of a facet the *front side* of the facet, and the area covered by that polygon

on the opposite side of the tetrahedron the *back side* (see Fig. 6). Note that this is an arbitrary terminology which is not necessarily related to their visibility order.

The vertex attributes include: the 3D position on the front side in eyespace coordinates (required in perspective projection only), the tetrahedron index (used later for reading the coefficients), the front-to-back intersection length (used in orthographic projection only), and two sets of barycentric coordinates, for the front side and the back side of the facet.

Under orthographic projection, the barycentric coordinates on the front and back sides of the facets and the thickness are interpolated linearly in *screen space*, and so we can pass them as vertex attributes as we have computed them. Under perspective projection, however, the following adjustment is required: The barycentric coordinates on the front side, $\mathbf{u}_f$, are interpolated correctly by the graphics hardware. The barycentric coordinates on the back side, $\mathbf{u}_b$, and the thickness, are computed by the fragment program according to following algorithm.

1. Let $\mathbf{f}$ be the thick vertex on the front side, and $\mathbf{b}$ its counterpart on the back side (see Fig. 6). From (2) and (8), we know α so that $\mathbf{b} = \alpha\mathbf{f}$.
2. Let $\mathbf{u}'_b = \alpha^{-1}\mathbf{u}_b$. Assign $\mathbf{u}'_b$ as a "back-side coordinates" vertex attribute of $\mathbf{f}$. Note that $|\mathbf{u}'_b| = \alpha^{-1}$. At the boundary, the front and back sides merge and $\alpha = 1$; therefore, equal values are assigned to both front-side and back-side coordinate attributes of the boundary vertices (one 1 and three 0's).
3. $\mathbf{u}'_b$ is interpolated correctly on the front side of the facet. In the fragment program, compute $\alpha = |\mathbf{u}'_b|^{-1}$. Recover $\mathbf{u}_b = \alpha\mathbf{u}'_b$.
4. Assign the *eyespace position* $\mathbf{p}_0$ of each vertex of the facet as a vertex attribute. $\mathbf{p}_0$ is interpolated correctly on the front side of the facet.
5. In the fragment program, compute $\mathbf{p}_1 = \alpha\mathbf{p}_0$. $\mathbf{p}_1$ is the point on the back side of the facet, projected on $\mathbf{p}_0$. The intersection length is now given by $\|\mathbf{p}_1 - \mathbf{p}_0\|$.

The proof of this algorithm is given in [1]. Note that all the attributes are continuous across the facets, which enables rendering the tetrahedron as a triangle fan. For comparison, in [1], we used the normal of the back side of the facet to compute the intersection length. As the normal is not a continuous attribute, we had to render the facets as independent triangles. The use of continuous attribute values surely improves the algorithm presented here over its predecessor, due to better vertex-cache coherence and reduction in the amount of data sent to the GPU.

5.3 The Complete Integral

Recall from (14) that the complete integral over a tetrahedron is the sum of basis function integrals multiplied by the density coefficients of the cell. Combinatorial analysis shows that the number of terms in a barycentric Bernstein multivariate polynomial of n variables and degree d has $n_f = \binom{n+d-1}{n-1}$ terms. In our case, $n = 4$, and for $d = 0, 1, 2, 3, 4$, we have $n_f = 1, 4, 10, 20, 35$.

To pass these coefficient to the fragment program, we use a 4-channel 32-bit float precision texture object and store the coefficients of the jth cell in consecutive texels. The fragment program then performs $n_r = \lceil \frac{n_f}{4} \rceil$ texture read operations.

We generate the fragment program *programmatically* by the main application. The fragment code generator enumerates the terms of a polynomial of degree d and expands their integral formula. Note here, that the order of enumeration is identical to the order of the coefficients, and both are based on the powers in the index $\mathbf{k}$, which relate to the barycentric coordinates computed from the vertices in their original winding order. Recall that we base our tessellation algorithm on the rendering order of the vertices, which may be different from their original order. The barycentric coordinates of all the facet vertices must be given with respect to the original order. The reason we keep track of the swapping of $\mathbf{v}_0$ and $\mathbf{v}_{j_0}$ is to be able to swap the barycentric coordinates back to the original order.

The density coefficients are derived by least-squares fitting of a polynomial to a volumetric data in the form of a CT data set. Typically, the CT numbers are between 0 and 4,095, and the density coefficients are on the same order. This creates a dynamic range issue. We accumulate the integrals to a 16-bit floating point pbuffer using an additive blending function, yet that may be insufficient, as the final integral values may be on the order of 10^5 or more, which overflows the 16-bit float. We resolve this using the following options:

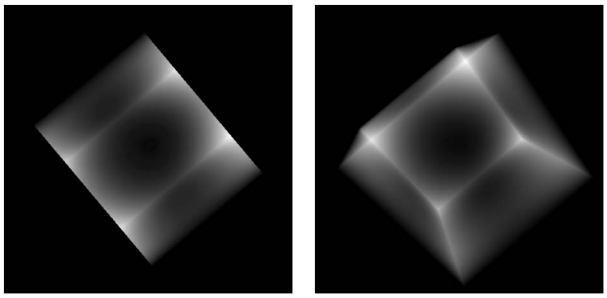
- Scale the model isotropically so that it fits inside a 2-unit cube. This is equivalent to scaling all the final integrals by the geometric scale factor of the mesh. Therefore, to restore the correct integral values, the pbuffer contents must be divided by the same scale factor used for the geometry.
- Scale and bias the density coefficients to a prescribed range, typically 0 to 1. To restore the correct integral values, the pbuffer contents must be divided by the density scale factor, and biased in proportion to the *total thickness* along the line of sight. Therefore, our fragment program outputs the thickness in the alpha channel, and it gets accumulated in parallel to the computed integral values.

6 Experiments and Results

The experiments and results presented here assess the quality of the rendered image, the performance of the algorithm, and its applicability in deformable registration.

6.1 High-Order Functions and Projection Models

Fig. 7 shows a rendering of a cube, which is made of five tetrahedral cells. Each cell has a third order density polynomial, with nonuniform coefficients arranged to produce high density near the vertices and a hole in the middle. We show an orthographic projection and a perspective projection with the same view transform, to demonstrate the difference between the projection models. Another example, showing the inner tetrahedron of this cube, with a similar density function, is in Fig. 1.



(a) (b)

Fig. 7. Rendering a cube with a nonuniform third-order density function. (a) Orthographic projection. (b) Perspective projection.

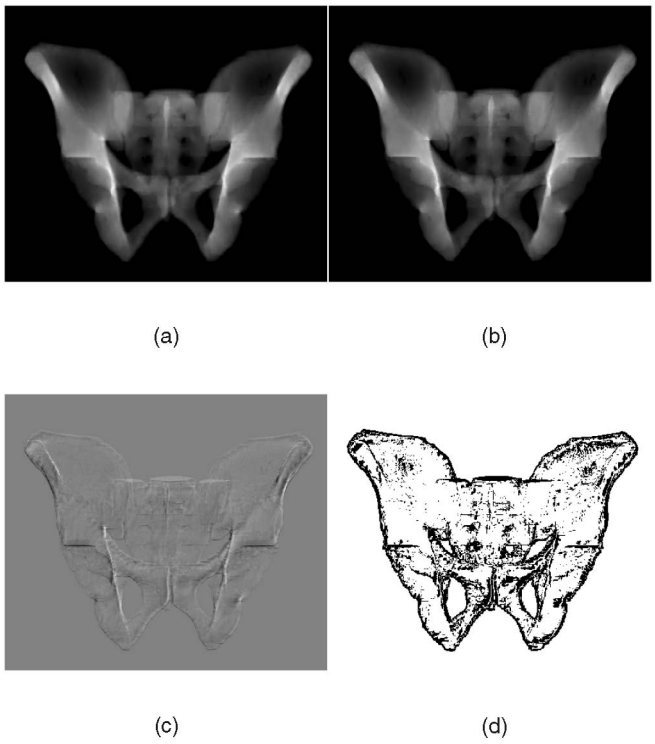
6.2 Model Creation

To explain the results involving anatomical models, we briefly present as background the model creation process, which generally follows Yao's work [4]. The tetrahedral mesh is part of an anatomical atlas, which includes a "mean" shape (the mesh), shape variation information, and density functions. The process is as follows:

1. Select a "master" CT data set, CT_0.
2. Manually label the voxels of the anatomy of interest in CT_0 (segmentation).
3. Create a "master" tetrahedral mesh, M_0, that covers the labeled voxels. We use a method developed by Mohamed and Davatzikos [17]. The spatial resolution of the mesh is defined by the user, and typically every cell contains many CT voxels. The mesh has n_v vertices and n_t tetrahedra.
4. For each instance CT_i in "subject" data sets, find an elastic transformation D_i that maps the voxels of CT_0 to corresponding voxels in CT_i. We use the method developed by Ellingsen and Prince [18].
5. Apply the deformation D_i to the vertices of M_0 to obtain a "subject" mesh M_i. Note that all the meshes created this way have identical graphs.
6. Take all the mesh instances as shape vectors of size $3n_v$, including M_0, and perform *principal component analysis* (cf. [19]) to obtain a mean shape $\bar{M}$ and *variation modes* $\{Y_i\}$ of the shape distribution. The variation modes are applied to the mean shape with different weights to create shape instances.
7. In each mesh instance M_i, for each tetrahedron $\mathbf{T}_j$, find polynomial coefficients $\{\beta^j_{\mathbf{k}}\}$ to approximate the density values in CT_i in the area occupied by $\mathbf{T}_j$. Store all the coefficients as a matrix Γ_i.
8. In the current implementation of the atlas, we use the mean of the density functions $\{\Gamma_i\}$ as the atlas density.

6.3 Comparison with Voxel DRR

Direct comparison between a CT data set and a tetrahedral model is difficult. The greatest differences are expected near the boundaries of the mesh, where there is no complete overlap between the cells of the mesh and the voxels of the CT. The quality of approximation of the CT data by the mesh depends on factors such as the cell sizes, sampling resolution and the degree of the density polynomials, which are beyond the scope of this paper.



(a) (b)

(c) (d)

Fig. 8. Comparing DRRs from a segmented pelvis CT and a tetrahedral mesh. (a) Anterior-posterior (AP) view of CT DRR. (b) AP view of mesh DRR with third order density functions. (c) Signed difference image (b)-(a). Zeros are mapped to the gray in the background, positive differences are bright, and negative differences are dark. (d) Locations of relative difference $|E(\mathbf{p})| \geq 0.05$.

We can therefore provide only a qualitative comparison between our model and a "ground-truth" CT. This is done by selecting only those CT voxels which are covered by mesh cells, and visualizing them using the Take package [20], then comparing the images with DRRs of our mesh.

An example comparison is given in Fig. 8. Fig. 8a shows a DRR of a "master" CT data set. Fig. 8b shows a DRR generated from a "master" model (without statistical processing), with $n_v = 14,375$, $n_t = 52,837$ and third order density functions. The signed-difference Fig. 8b - Fig. 8a is shown in Fig. 8c, with zero differences mapped to the gray in the background, positive differences brighter, and negative differences darker. The areas of greatest differences, both positive and negative, are near boundaries. Fig. 8d shows a relative pixelwise error measure: $E(\mathbf{p}) = \frac{I_{Model}(\mathbf{p}) - I_{CT}(\mathbf{p})}{I_{CT}(\mathbf{p})+1}$, thresholded at $|E(\mathbf{p}) >= 0.05|$ to show the locations of relative error greater than 5 percent as black pixels. Data about the dynamic ranges in the images is in Table 1.

6.4 Contribution of Higher-Order Functions

To understand the impact of higher-order density functions on the resulting image, we compute the difference $I_d - I_{d-1}$, with I_d being an image generated with polynomials of degree d. Example results are given in Fig. 9. Numerical data about the differences is in Table 2. The table shows the maximum and minimum intensity values over a trajectory

TABLE 1
Dynamic Ranges in Fig. 8

Image	(a)	(b)	(c)
min	0.00	-39.20	-26,358.01
max	129,877.65	134,750.91	25,648.44

of 25 images, the maximum and minimum pixel differences between the images of degree d and the images of degree $d-1$, and the root of the mean square (RMS) difference. As we can see visually and numerically, the differences are diminishing when the degree is increased. However, the dynamic range of the images increases with the degree, and begins to include negative values. These are likely to be the result of polynomial overfitting.

6.5 Performance

To evaluate the performance of the algorithm, we tested the pelvis mesh with different density function degrees. We fit polynomials of degrees 0 to 4 to the cells of the mesh. The model was imaged four times over a camera trajectory of 25 frames in 9° increments (a 216° arc). The performance results in two different computers are summarized in Table 3, which also includes the machine specifications. The computer denoted as *CPU-1/GPU-1* was also used for performance measure in [1].

Comparing these results with [1], we observe a speedup of more than nine times in the tessellation done on the CPU. We attribute the speedup to multiple factors, including:

1. a reduction in the number of vertices sent per tetrahedron from 9-12 to 5-6 (for 3-4 tessellated facets), due to the use of a triangle fan;
2. a reduction in the size of vertex attributes from 20 floats to 12;
3. a simpler and more efficient tessellation algorithm; and
4. moving the computation of view and projection transforms for the vertices from the CPU, as part of tessellation, to a GPU vertex program.

It is hard to determine the exact contribution of each factor, but all these improvements became possible with the new tessellation algorithm presented here. There is a smaller speedup, about 32 percent on average, in the OpenGL rendering time for degrees 0 to 3, which we attribute mostly to the use of a triangle fan. Note that the rendering of the fourth order polynomials improves most significantly with the transition from GPU-1 to the newer GPU-2, which we attribute to improved hardware capacities.

TABLE 2
Numerical Data for Fig. 9

d	**0**	**1**	**2**	**3**	**4**
max	233,942	234,778	236,507	237,882	240,424
min	0	0	0	-43	-85
max dif	–	16,336	8,339	3,653	3,598
min dif	–	-10,140	-6,506	-3,237	-2,984
rms dif	–	405	259	119	104

The differences are computed as $I_d - I_{d-1}$ and are not defined for $d = 0$.

6.6 Deformable 2D-3D Registration

To demonstrate the use of our method in a clinically-oriented application, we present here results of a deformable 2D-3D registration experiment between a statistical atlas of the pelvis and DRR's generated from a CT scan. We created an atlas with the average shape and statistics of 13 subjects ($n_v = 16,487$, $n_t = 62,541$). We created DRRs from one of the subject datasets from two orthogonal views to be used as targets. Then, we matched the atlas to the target images by maximization of mutual information. We search for a rigid transformation and a set of weights applied to the deformation modes of the atlas.

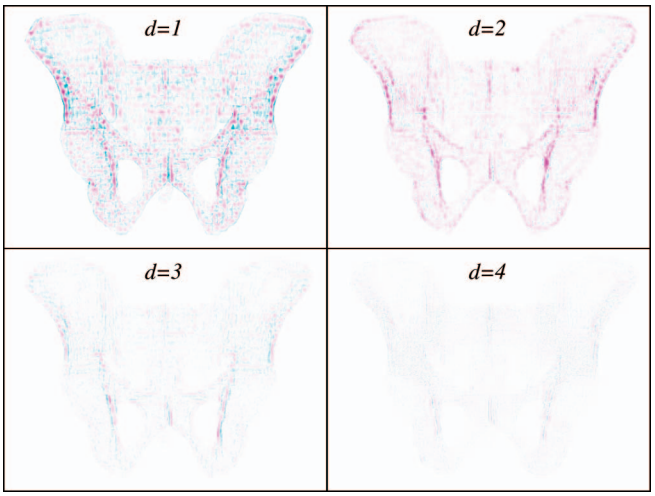


Fig. 9. Difference between images of the pelvis mesh with polynomials of degree d and degree $d-1$. The images show *signed differences*, with negative values in cyan, zeros in white, and positive differences in magenta. The values of the differences are scaled equally in all four images. d goes from 1 to 4.

TABLE 3
Comparing the Performance of Rendering Frames of the Pelvis Mesh with Different Polynomial Degrees

Degree	**0**	**1**	**2**	**3**	**4**
CPU-1 (ms/frame)	30.1	30.7	30.7	30.9	31.3
GPU-1 (ms/frame)	15.7	19.8	44.4	120.3	1279.2
CPU-2 (ms/frame)	26.1	25.8	25.0	26.1	27.8
GPU-2 (ms/frame)	7.6	8.0	24.3	81.9	298.3
Instructions ($n_i(d)$)	34	44	101	275	749
Terms (n_f)	1	4	10	20	35
Texture reads : $\lceil n_f/4 \rceil$	1	1	3	5	9

CPU times involve computing the tessellation and vertex attributes. GPU times involve sending the triangle fans to OpenGL and computing the line integrals in the fragment program. n_i is the size of the compiled Cg fragment program using fragment profile fp30. n_f is the number of terms (and coefficients) in a Bernstein polynomial of degree d. $\lceil n_f/4 \rceil$ is the number of texels required to store n_f coefficients.
The rendering times are means over 100 frames. Image size = 512^2 pixels. Operating system: Windows XP.
CPU-1: dual Xeon PC, 2.80GHz. GPU-1: NVidia GeForce 6800 GT, AGP 8X. CPU-2: Pentium Extreme Edition Dual Core with Hyper Threading Technology (3.20 GHz). GPU-2: Nvidia GeForce 7800 GTX.

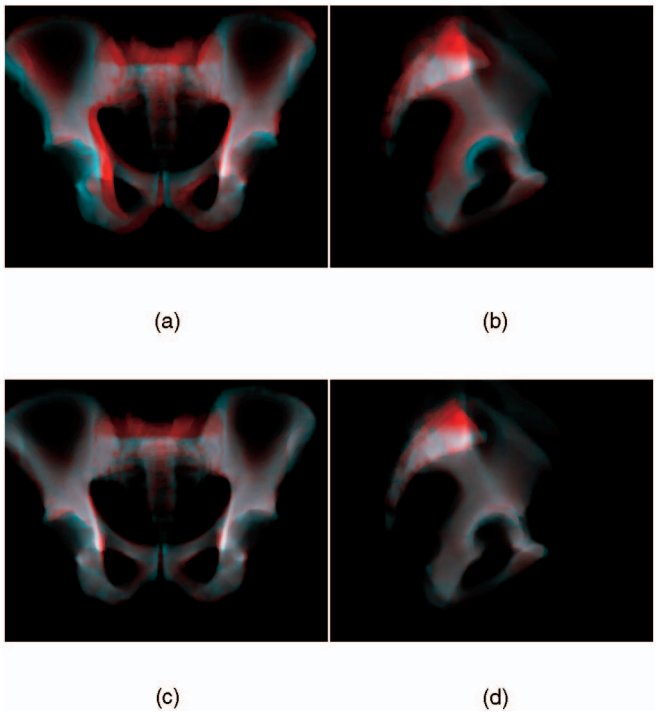


Fig. 10. Results of deformable 2D-3D registration using the mesh rendering algorithm. The model is in red overlaid on the target image in cyan. (a) and (b) show an anterior-posterior and lateral view of the best rigid registration. (c) and (d) show the best deformable registration.

Fig. 10 shows an overlay of the transformed model image in red and the target DRR image in cyan. The top row shows the best result attained using a rigid transformation only. We can see areas of mismatch as colored regions. The bottom row shows the best result attained when deformations were included. Clearly, there is an improvement when deformable registration is used. Note that this is not a leave-out validation, as the subject used as target was included in the statistics. Nevertheless, this demonstrates the potential of using the atlas, and the impact of a fast rendering method.

6.7 Additional Anatomical Models

Fig. 11 shows DRR images of a mesh generated from a CT scan of a knee phantom. The number of vertices is 11,704; the number of tetrahedra is 46,020. Some fine details can be seen, such as the difference between the cortical bone and the canal in the tibia and the fibula in Fig. 11a. Since the cells were not specifically aligned with tissue boundaries, this demonstrates again the contribution of higher-order density functions.

7 Discussion and Future Work

7.1 Performance and Quality of the Rendering

To analyze the results of the rendering algorithm, we need to consider both the execution time of the algorithm and the quality of the result. Observing the results in Table 3 and Table 2, we can draw the following notions.

- The computation time for the tessellation algorithm and vertex attributes is practically the same, regardless of the degree, as one may expect.

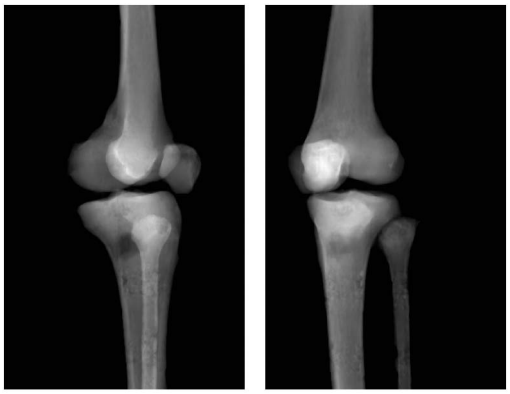


Fig. 11. DRR images of a tetrahedral model of bones segmented from a knee phantom CT scan, including: femur, patella, tibia, and fibula. Two views are shown. The images were created using our algorithm.

- The size of the integral expression (14), which is proportional to the number of instructions in the fragment program ($n_i(d)$), increases rapidly with the degree of the polynomial. We do not have yet an expression for the *size* of the formula, but at least in this data, it shows a trend of exponential growth: $\log(n_i(d))$ is approximately linear with respect to d.
- The relative contribution of increasing the degree of the polynomial to the detail of the image diminishes, as can be seen in Fig. 9 and Table 2.

These results suggest that we may need to modify our method if we want to develop an efficient rendering method for higher polynomial degrees. Several possible improvements are:

- Try to simplify and optimize the integral formula, for example by caching values of low-order terms from (14). This requires a more sophisticated generator of the fragment program source code, and a better understanding of the structure of the formula.
- Pipeline the CPU and GPU computation stages.
- Encode the tessellation algorithm in a vertex program.

Our results suggest that in terms of precision, third order functions are probably sufficient for the models we tested, while not adding a heavy performance cost. However, the relation between the geometrical level of detail (that is, the number and size of the cells in the mesh), the degree of the polynomials, and the functional approximation quality (e.g., the residual error between the intensity values in a CT study and our model), requires further study. Other uses of the statistical atlas, such as approximated tomographic reconstruction, may dictate more constraints on the level of detail we can use.

7.2 Deformations, Intensity Variations, and Registration

We have successfully incorporated our rendering method in a deformable registration framework. Applying a deformation to the mesh requires repositioning the model vertices and recomputing the inverse vertex matrices for the

tetrahedra. A few tests we performed show that this does not decrease the performance in a significant way.

Further development of the rendering algorithm can include applying the deformations in GPU memory, instead of the main memory and CPU operations in our current code. This requires additional programming, including the encoding of the tessellation in a vertex program, and inverting 4×4 matrices on the GPU.

Deformations applied to a tetrahedral mesh may cause self intersections between topologically-adjacent cells, and also between topologically-remote cells, as the mesh may fold over itself. We apply deformations in the atlas construction phase, and in the deformable registration. Part of our ongoing research is on resolving self-intersection issues. We believe that the inverse vertex matrix $M_{\mathbf{T}}^{-1}$ and barycentric coordinates can help in parts of this problem, as well as in the rendering algorithm we present.

To enhance the approximation quality between our atlas and real data, we are planning to include intensity variations in the atlas, using similar analysis as we have for the shape variations. Since the density coefficients are stored in a texture object, it should be possible to change the density functions through texture operations.

The current 2D-3D registration requires reading the rendered image back to the main memory in order to compute the similarity measure between it and the target image. To reduce the performance bottleneck of the read-back operation, we are looking for ways to close the optimization loop by computing the similarity measure on the GPU. Then, a complete 2D-3D registration framework can be established using GPU operations almost everywhere.

7.3 Additional Projection Models

One projection model that is sometimes applicable to medical imaging techniques is known as a "linear pushbroom" model (see [21], for example). It models the imaging process of a flat fan-beam moving along a line, and is therefore different from both perspective (cone-beam) and orthographic projections. One may regard it as an in-between model, with perspective foreshortening along one axis of the image, and no foreshortening along the other. Two examples of imaging systems that use this model are: CT scout images and dual x-ray imaging.

As future research, we would like to develop a similar rendering solution, with tessellation and interpolation, under the pushbroom projection model.

8 Conclusion

We have presented a novel algorithm for computing DRR images of an unstructured grid, with higher-order attenuation functions. Our method uses a closed formula for integrating the function along the line of sight. In this paper, we have described a simple and efficient algorithm for computing the silhouette of a projected tetrahedron and its vertex attributes. We see this as a significant algorithmic development over older works on projected tetrahedra, many of which contain inaccuracies with regards to perspective projection.

The experimental results presented here show that using higher-order attenuation functions influences the quality of the final image, and potentially improves it; but this contribution comes with a price: The size of the current formulation of the integral shows a trend of exponential growth with the degree, which impacts the performance. These problems should be addressed in future research. Even so, our method is at least two orders of magnitude faster than an older, CPU-only method we used before its development.

To conclude, we consider the geometric solutions for the silhouette and the interpolation of thickness and barycentric coordinates presented here to be simple and elegant. The ability to include shape and density variations in the rendered image is a key element in studying and using generic anatomical models, created from a population study. We consider this a leap forward in the application environment of computer assisted surgery.

Acknowledgments

The authors would like to thank the following people for their help in creating the statistical pelvis atlas: Gouthami Chintalapani, Lotta Ellingsen, Ashraf Mohamed, Pauline Pelletier, and Na Song. They would also like to thank the reviewers of this paper for their comments, which helped them improve the text, and motivated them to improve the results. The pelvis CT data sets were provided by Dr. Ted DeWeese and Dr. Lee Myers. This work was supported in part by US National Science Foundation ERC Grant EEC9731478, by NIH/NIBIB research grant R21-EB003616, and by US National Science Foundation ITR Grant AST-0313031.

References

[1] O. Sadowsky, J.D. Cohen, and R.H. Taylor, "Rendering Tetrahedral Meshes with Higher Order Attenuation Functions for Digital Radiograph Reconstruction," *Proc. Conf. IEEE Visualization,* pp. 303-310, 2005.

[2] D. Laney, S.P. Callahan, N. Max, C.T. Silva, S. Langer, and R. Frank, "Hardware Accelerated Simulated Radiography," *Proc. Conf. IEEE Visualization,* pp. 343-350, 2005.

[3] P. Shirley and A. Tuchman, "A Polygonal Approximation to Direct Scalar Volume Rendering," *Proc. Workshop Volume Visualization,* 1990.

[4] J. Yao, "A Statistical Bone Density Atlas and Deformable Medical Image Registration," PhD dissertation, Johns Hopkins Univ., 2002.

[5] M. Kraus, W. Qiao, and D.S. Ebert, "Projecting Tetrahedra without Rendering Artifacts," *Proc. Conf. IEEE Visualization,* pp. 27-33, Oct. 2004.

[6] B. Wylie, K. Moreland, L.A. Fisk, and P. Crossno, "Tetrahedral Projection Using Vertex Shaders," *Proc. 2002 IEEE Symp. Volume Visualization and Graphics,* pp. 7-12, 2002.

[7] M. Levoy and P. Hanrahan, "Light Field Rendering," *Proc. SIGGRAPH '96,* pp. 31-42, 1996.

[8] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen, "The Lumigraph," *Proc. SIGGRAPH '96,* pp. 43-54, 1996.

[9] D.A. LaRose, "Iterative X-Ray/CT Registration Using Accelerated Volume Rendering," PhD dissertation, Carnegie Mellon Univ., 2001.

[10] S. Röttger, M. Kraus, and T. Ertl, "Hardware-Accelerated Volume and Isosurface Rendering Based on Cell-Projection," *Proc. Conf. IEEE Visualization,* pp. 109-116, 2000.

[11] M. Weiler, M. Kraus, and T. Ertl, "Hardware-Based View-Independent Cell Projection," *Proc. IEEE Volume Visualization and Graphics Symp.,* pp. 13-22, 2002.

[12] K. Moreland and E. Angel, "A Fast High Accuracy Volume Renderer for Unstructured Data," *Proc. IEEE Symp. Volume Visualization and Graphics,* pp. 9-16, Oct. 2004.

[13] P.L. Williams, "Visibility Ordering Meshed Polyhedra," *ACM Trans. Graphics,* vol. 11, no. 2, pp. 103-126, Apr. 1992.

[14] C.T. Silva, J.S. Mitchell, and P.L. Williams, "An Exact Interactive Time Visibility Ordering Algorithm for Polyhedral Cell Complexes," *Proc. IEEE Symp. Volume Visualization,* pp. 87-94, 1998.
[15] R. Farias, J. Mitchell, and C. Silva, "Zsweep: An Efficient and Exact Projection Algorithm for Unstructured Volume Rendering," *Proc. 2000 ACM/IEEE Volume Visualization and Graphics Symp.,* pp. 91-99, 2000.
[16] S. Callahan, M. Iktis, J. Comba, and C.T. Silva, "Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 3, May/June 2005.
[17] A. Mohamed and C. Davatzikos, "An Approach to 3D Finite Element Mesh Generation from Segmented Medical Images," *Proc. IEEE Int'l Symp. Biomedical Imaging (ISBI),* pp. 420-423, 2004.
[18] L. Ellingsen and J. Prince, "Mjolnir: Deformable Image Registration Using Feature Diffusion," *Proc. Conf. SPIE Medical Imaging,* 2006.
[19] T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham, "Active Shape Models—Their Training and Application," *Computer Vision and Image Understanding,* vol. 6, no. 1, pp. 38-59, 1995.
[20] J. Muller-Merbach, "Simulation of X-Ray Projections for Experimental 3D Tomography," Technical Report, Image Processing Laboratory, Dept. of Electrical Eng., Linkoping Univ., SE-581 83, Sweden, 1996.
[21] R. Gupta and R. Hartley, "Linear Pushbroom Cameras," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 9, pp. 963-975, Sept. 1997.

Ofri Sadowsky received the MSc degree in computer science at the Hebrew University in Jerusalem, Israel, in 2001. He is currently a PhD student in the Computer Science Department at The Johns Hopkins University. His past projects were in the areas of image registration and camera calibration in x-ray systems. His current research is on statistical anatomical models of bones. He is a student member of the IEEE.

Jonathan D. Cohen received the PhD degree in computer science from the University of North Carolina at Chapel Hill in 1998, and the BA degree in computer science and music from Duke University in 1991. He is on the computer science faculty of The Johns Hopkins University. His research interests include interactive 3D visualization, geometric algorithms, and graphics architecture.

Russell H. Taylor (M'76-F'94) received the BES degree from The Johns Hopkins University in 1970 and the PhD degree in computer science from Stanford University in 1976. Subsequently, he was a research staff member and manager at IBM Research until he moved to the Johns Hopkins University in 1995, where he is currently a professor of computer science, with joint appointments in mechanical engineering, radiology, and surgery. He is also director of the US National Science Foundation Engineering Research Center on Computer-Integrated Surgical Systems and Technology. His research interests include robotics, modeling, and computer-assisted surgery. He is a fellow of the IEEE.