Rendering Tetrahedral Meshes with Higher-Order Attenuation Functions for Digital Radiograph Reconstruction

Ofri Sadowsky Jonathan D. Cohen Russell H. Taylor*

Johns Hopkins University

ABSTRACT

This paper presents a novel method for computing simulated x-ray images, or DRRs (digitally reconstructed radiographs), of tetrahedral meshes with higher-order attenuation functions. 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. Our application of DRRs is in intra-operative "2D-3D" registration, i.e., finding the pose of the CT dataset given a small number of patient radiographs. We register 2D patient images with a *statistical* tetrahedral model, which encodes the CT intensity numbers as Bernstein polynomials, and includes knowledge about typical shape variation modes. The unstructured grid is more suitable for applying deformations than a rectilinear grid, and the higher-order polynomials provide a better approximation of the actual density than constant or linear models. The intra-operative environment demands a fast method for creating the DRRs, which we present here. We demonstrate this application through the creation and use of a deformable atlas of human pelvis bones.

Compared with other works on rendering unstructured grids, the main contributions of this work are: 1) Simple and perspective-correct interpolation of the thickness of a tetrahedral cell. 2) Simple and perspective-correct interpolation of front and back barycentric coordinates with respect to the cell. 3) Computing line integrals of higher-order functions. 4) Capability of applying shape deformations and variations in the attenuation function without significant performance loss. The method does not depend on for pre-integration, and does not require depth-sorting of the visualized cells.

We present imaging and timing results of implementing the algorithm, and discuss the impact of using higher-order functions on the quality of the result and the performance.

Keywords: 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 intra-operative "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 projection model, 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.

This paper presents a novel algorithm for generating DRRs of a tetrahedral mesh. Each cell contains a representation of the local values of the attenuation function as a higher-order Bernstein polynomial in barycentric coordinates. This algebraic representation yields a closed formula for computing the line integrals of the function. Our algorithm, based on the Projected Tetrahedra (PT) algorithm by Shirley and Tuchman [13], correctly interpolates the parameters of the integral formula, and uses the GPU fragment processors to compute it for each fragment.

The paper is organized as follows. Section 2 presents the context of this work in the frame of CAS and compares our work with related works on rendering unstructured grids. Section 3 presents the mathematical formulation of our algorithm, and explains the details of its implementation. Section 4 shows results of rendering experiments using the algorithm. We discuss the meaning of the results and potentials for future work on the algorithm in Section 5, and conclude in Section 6.

2 RELATED WORK

This work was developed in the context of a statistical atlas of bone anatomy, continuing the work of Yao [18]. In his 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. 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 actual x-ray images of a patient and dynamic DRRs of the statistical atlas. Yao implemented a DRR generator which used a numerical approximation of the integrals as sums of discrete samples of the attenuation function along the line of sight, in a manner similar to ray tracing. However, the use of polynomials provides a closed formula for the line integrals over the line segment that intersects 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 [13], is the basis to many works on rendering unstructured grids. The paper starts with a method to decompose a rectilinear voxel grid to a tetrahedral mesh. The rendering of individual tetrahedra is done by breaking them further into three or four triangular facets (tessellation), which are sent to the rendering pipeline. The integrand function is constant per cell (*piecewise constant*), and the thickness interpolation is linear, which is correct only under orthographic projection. Depth sorting is required as well. These weaknesses in the original PT algorithm call for the improvements that followed.

Wylie et al. [17] 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. They still use linear interpolation of the thickness property, which assumes an orthographic projection model.

A number of volume rendering methods use precomputed line integrals to accelerate the rendering. So-called "light field" meth-

*e-mail: ofri,cohen,rht@cs.jhu.edu

ods [7, 3] are one set of examples. In the particular context of DRR-based registration, LaRose [6] presented a rendering method that uses a 4-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. Röttger et al. [11] use precomputed integrals to render a tetrahedral mesh. The rendered function 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.

Weiler, Kraus, and Ertl [15] compute the entry and exit point of the ray to and from the cell under perspective projection by solving line-plane-intersection equations, a method also adopted by Moreland and Angel [9]. The main drawback of this method was that it was too complex to implement on a GPU fragment unit. Kraus et al. [5] followed by presenting a method to interpolate the thickness and intersection points under a perspective projection model, still using pre-integration. Their per-vertex data includes two sets of coordinates, for the front and back sides of the facet, which are interpolated on the GPU.

Many of the works cited above require depth sorting of the cells, using methods such as [16, 14, 2], because the volumetric function includes emission, which is attenuated only by cells lying in front of the current cell. The model of the rendered function in our paper is simpler than in some of these works, as it includes only attenuation and no emission or color. This is because we model x-ray imaging. However, the function is *piecewise polynomial*, which generalizes the linear model to a higher degree, and allows a compact approximated representation of complicated structures. Our algorithm is based on the PT tessellation, and improves the PT algorithm to interpolate the parameters required for evaluating the integral formula. The meshes we used were generated automatically from labeled images using an algorithm developed by Mohamed [8], yet our algorithm does not depend on the specific mesh generation method.

Compared to the works cited above on rendering unstructured grids, the main contributions and features of our algorithm are:

1. Simple and perspective-correct interpolation of the thickness of a tetrahedral cell (alternative to [5]).
2. Simple and perspective-correct interpolation of front and back barycentric coordinates with respect to the cell (compared to actual position in [5]).
3. Computing line integrals of higher-order functions.
4. Capability of applying shape deformations and variations in the attenuation function without significant performance loss.
5. No need for pre-integration.
6. No need for depth-sorting of the visualized cells.

These properties are especially important within the context of deformable "2D-3D" registration of x-rays to generic anatomical templates. The last property results from the cumulative nature of the attenuation model, which enables us to use OpenGL blending for summing up the results into a high-dynamic-range frame buffer. We currently implement the PT tessellation to run on the CPU, but the algorithm we propose may be usable in conjunction with a GPU vertex program such as Wylie's.

3 Algorithm

3.1 Background

Our imaging model is based on the attenuation formulas known as Beer's law. For a point $\mathbf{p}$ in space, the x-ray flux I_{out} leaving in a certain direction is equal to $\alpha(\mathbf{p}) \cdot I_{in}$ for the same direction, with $0 \le \alpha(\mathbf{p}) \le 1$. Along a parametric line $\mathbf{p}(t)$, the overall attenuation of energy is henceforth given by the formula

$$I_{out} = I_{in} \cdot e^{-\int_0^1 \mu(\mathbf{p}(t))dt} \tag{1}$$

where $\mu(\mathbf{p}) = -\ln \alpha(\mathbf{p})$ is called the *linear attenuation coefficient*.

The volumetric attenuation function in our project is represented by a tetrahedral mesh with barycentric Bernstein polynomials encoding the local density in each cell. For a tetrahedral cell $\mathbf{T}_j$, the structure of the polynomial is

$$f^j(\mathbf{u}) = \sum_{|\mathbf{k}|=d} \beta_{\mathbf{k}}^j B_{\mathbf{k}}^d = \sum_{|\mathbf{k}|=d} \beta_{\mathbf{k}}^j \binom{d}{\mathbf{k}} u_0^{k_0} u_1^{k_1} u_2^{k_2} u_3^{k_3} \tag{2}$$

where $\mathbf{u} = (u_0, u_1, u_2, u_3)^T$ are barycentric coordinates ($u_0 + u_1 + u_2 + u_3 = 1$) with respect to $\mathbf{T}_j$; $\mathbf{k} = (k_0, k_1, k_2, k_3)$ is the *power index* of a term, with $d = |\mathbf{k}| = k_0 + k_1 + k_2 + k_3$ 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. Since the collection of barycentric Bernstein basis functions $\{B_{\mathbf{k}}^d\}$ is fixed, and the degree of the function is equal in all the cells in our model, we only need to store the free coefficients for each cell to recover the values of the attenuation function everywhere.

The transformations between Cartesian coordinates and barycentric coordinates with respect to $\mathbf{T}_j$ are linear, and given by

$$\begin{aligned} \mathbf{p}_{(h)} &= M_{(h)}(\mathbf{T}_j)\mathbf{u}, \\ \mathbf{u} &= M_{(h)}(\mathbf{T}_j)^{-1}\mathbf{p}_{(h)} \end{aligned} \tag{3}$$

We use the subscript (h) to denote the straightforward conversion to homogeneous coordinates by adding 1 as a fourth component. $\mathbf{p}$ and $\mathbf{u}$ are Cartesian and barycentric coordinates, respectively. $M_{(h)}(\mathbf{T}_j)$ is a 4×4 matrix, whose columns contain the (homogeneous) Cartesian coordinates of the vertices of the tetrahedron.

The DRR image of *one tetrahedron* is computed by taking the exponential of line integrals of the attenuation function. The *final* DRR image is computed by adding together the line integrals from all the tetrahedra, and taking the exponential of the sum. In this paper, we will regard the exponential taking as a post-processing step, which can be applied immediately before displaying or saving the image, and focus on computing and summing the integrals. The line integral of a single Bernstein basis function $B_{\mathbf{k}}^d(\mathbf{u})$ between the Cartesian points $\mathbf{p}_0$ and $\mathbf{p}_1$, with corresponding barycentric coordinates $\mathbf{u}_0, \mathbf{u}_1$, is given by the formula

$$\begin{aligned} \int_{\mathbf{p}_0}^{\mathbf{p}_1} B_{\mathbf{k}}^d(\mathbf{u})d\mathbf{u} &= \|\mathbf{p}_1 - \mathbf{p}_0\| \int_0^1 B_{\mathbf{k}}^d(\mathbf{u}(t))dt \\ &= \frac{\|\mathbf{p}_1 - \mathbf{p}_0\|}{d+1} \sum_{\mathbf{q} \sqsubseteq \mathbf{k}} B_{\mathbf{q}}^{|\mathbf{q}|}(\mathbf{u}_0) B_{\mathbf{k}-\mathbf{q}}^{|\mathbf{k}-\mathbf{q}|}(\mathbf{u}_1) \end{aligned} \tag{4}$$

where $\mathbf{u}(t) = \mathbf{u}_0 + t(\mathbf{u}_1 - \mathbf{u}_0)$. We use the notation $\mathbf{q} \sqsubseteq \mathbf{k}$ to denote that $q_i \le k_i$ for all i. Note that the integral is computed from the barycentric coordinates $\mathbf{u}_0, \mathbf{u}_1$, the degree d, and the length of the line segment $\|\mathbf{p}_1 - \mathbf{p}_0\|$. These parameters will be required in our fragment program. We will refer to the product $B_{\mathbf{q}}^{|\mathbf{q}|}(\mathbf{u}_0) B_{\mathbf{k}-\mathbf{q}}^{|\mathbf{k}-\mathbf{q}|}(\mathbf{u}_1)$ as the inner low-order term product in the integration formula. The actual development of this formula is beyond the scope of this paper.

3.2 Classifying and tessellating projected tetrahedra

As described in [13], the silhouettes of projected tetrahedra can be classified in four classes. Each class defines a tessellation of the

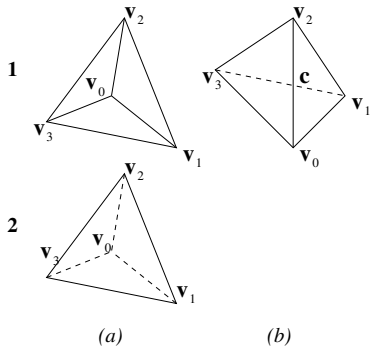

Figure 1: Classification of projected tetrahedra. (a) A three-facet projection. (b) A four-facet projection. The two (a) images indicate: (1) $\mathbf{v}_0$ is pointing towards the observer; (2) $\mathbf{v}_0$ is pointing away from the observer.

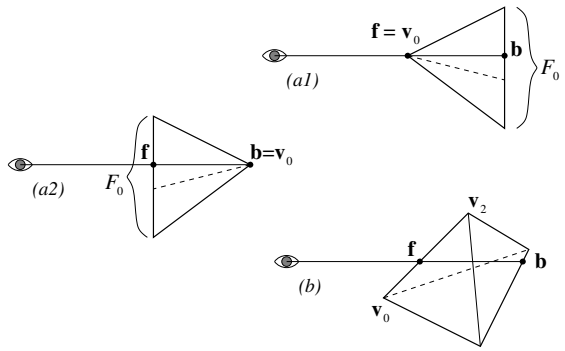

Figure 2: Side view of projected tetrahedra, indicating the near thick point **f** and the far thick point **b**, which are projected along the same line of sight. The labels (a1), (a2), and (b) correspond to the labels in Figure 1.

silhouette into smaller *facets*, whose number is one to four. To simplify the analysis and the implementation (cf. [17]), we use only two of these classes: three and four facets, as illustrated in Figure 1, and gather the degenerate cases into the three-facet class (a).

The tessellation procedure involves the following steps:

1. Determine the class of the silhouette: triangle or quadrilateral.
2. Find the thick point, which will be a vertex shared among all the facets.
3. Compute the vertex attributes for the vertices of the facets.

To determine the class, we project all the vertices of the tetrahedron to the imaging plane, and then compute line intersections between three pairs of projected non-adjacent edges. The intersection may not exist if the shape happens to be a quadrilateral with parallel edges, but in the other cases it may occur inside or outside of an edge. We currently implement these tests in a CPU module, which allows for optimizations such as early exit. For example, if we find an intersection that is inside both edge projections, we determine that it's a quadrilateral, and skip the other tests.

After the case is determined, we reorder the vertices as in Figure 1. This simplifies the following steps in the algorithm, as we can apply a predefined set of operations on the vertices, and retain numerically consistent results. However, it is essential to keep track of the vertex permutation, since the barycentric coordinates and the polynomial terms are ordered based on the vertex order given originally in the mesh. For the simplicity of the discussion that follows, we will assume that no reordering is necessary.

For the thick point, we need to determine a front point **f** and a back point **b** (see Figure 2), which are both projected to the same point **c** on the image plane. In the triangle case, **c** coincides with the projection of $\mathbf{v}_0$, and **f** and **b** are either equal to $\mathbf{v}_0$ or to a point on the opposite face F_0, found by intersecting the line of sight of $\mathbf{v}_0$ with the face F_0. In the quadrilateral case, **c** is found by intersecting the edges between the projections of $(\mathbf{v}_0,\mathbf{v}_2)$ and $(\mathbf{v}_3,\mathbf{v}_1)$, and then back-projected to a line of sight $l_\mathbf{c}$. The intersection of $l_\mathbf{c}$ with $(\mathbf{v}_0,\mathbf{v}_2)$ in eye-space is the point **f**, and the intersection with $(\mathbf{v}_3,\mathbf{v}_1)$ is the point **b**.

Finally, we define the facets using the front thick vertex **f** and the outer vertices of the tetrahedron. In a triangle silhouette, the facets are: $(\mathbf{f},\mathbf{v}_1,\mathbf{v}_2),(\mathbf{f},\mathbf{v}_2,\mathbf{v}_3),(\mathbf{f},\mathbf{v}_3,\mathbf{v}_1)$. In a quadrilateral, they are:$(\mathbf{f},\mathbf{v}_0,\mathbf{v}_1)$, $(\mathbf{f},\mathbf{v}_1,\mathbf{v}_2),(\mathbf{f},\mathbf{v}_2,\mathbf{v}_3),(\mathbf{f},\mathbf{v}_3,\mathbf{v}_0)$.

3.3 Vertex attributes, fragment attributes, and their interpolation

As noted in Equation (4), computing the line integral for a given fragment, covered by a facet $\mathbf{F}_i^j$ of the tetrahedron $\mathbf{T}_j$, requires obtaining the barycentric coordinates $\mathbf{u}_f$ and $\mathbf{u}_b$ of the intersections between the line of sight and the front and back faces of $\mathbf{T}_j$, and the distance between these two intersection points. In this part, we derive formulas for simple per-fragment interpolation of these attributes from vertex attributes which are assigned to the facet. The final computation of the integral parameters is done in the fragment program.

We start by noting that as a by-product of computing the line-plane or line-line intersections for the points **f** and **b** we obtain the components of their barycentric coordinates: $\mathbf{u}(\mathbf{f})$ and $\mathbf{u}(\mathbf{b})$. They are included in the unknown variables of the intersection equations. We also use the fact that at the tetrahedral vertex $\mathbf{v}_i$ the barycentric coordinate u_i is 1, and the other three are zero, and that if a point is on the face F_k, its barycentric coordinate u_k is zero. Therefore, the tessellation algorithm already gives all the attributes needed to interpolate the barycentric coordinates. Next, we show how these are interpolated per fragment.

Front barycentric coordinates $\mathbf{u}_f$. Given the front barycentric coordinates at the vertices of the facet, the coordinates at the fragment are obtained by a simple linear interpolation. Therefore, just by passing values of $\mathbf{u}_f$ at the vertices as texture coordinates, we guarantee perspective-correct interpolation per fragment.

Back barycentric coordinates $\mathbf{u}_b$. Fragment attributes are interpolated based on the eye-space coordinates on the *front side* of the facet. Therefore, we have to base the interpolation of the *back side* barycentric coordinates on an attribute of the front side. Let F_b be the back side of $\mathbf{F}_i^j$, defined by the vertices $(\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3)$. Let F_f be the front side for the same facet. A point $\mathbf{x}\in F_b$ can be expressed using the convex combination of the vertices:

$$\begin{aligned}\mathbf{x} &= \begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3\end{bmatrix}(w_1,w_2,w_3)^T = M_{F_b}\mathbf{w}_{F_b}(\mathbf{x})\\ \mathbf{w}_{F_b}(\mathbf{x}) &= M_{F_b}^{-1}\mathbf{x}\end{aligned} \tag{5}$$

with $|\mathbf{w}_{F_b}(\mathbf{x})| = w_1+w_2+w_3 = 1$. Note that $\mathbf{w}_{F_b}(\mathbf{x})$ contains the *nontrivial barycentric coordinates* of $\mathbf{x}$ on the face F_b, i.e., those components which are not trivially zero.

If $\mathbf{y}\in F_f$ is given in eye-space coordinates, the point on F_b which is projected on $\mathbf{y}$ is $\mathbf{x} = s\mathbf{y}$ for some $s>1$. The nontrivial barycentric coordinates of $\mathbf{x}$ are:

$$\mathbf{w}_{F_b}(\mathbf{x}) = M_{F_b}^{-1}\mathbf{x} = sM_{F_b}^{-1}\mathbf{y} = sM_{F_b}^{-1}M_{F_f}\mathbf{w}_{F_f}(\mathbf{y}) \tag{6}$$

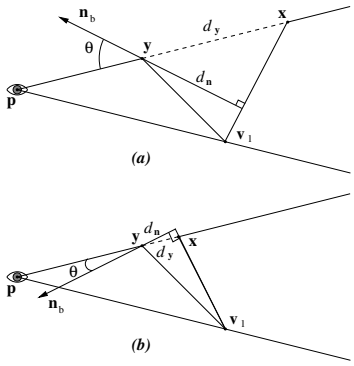

Figure 3: Interpolating the thickness along the line of sight. Here, $\mathbf{p}$ is the eye position; $\mathbf{y}$ a point on the front face, and $\mathbf{x}$ is the corresponding point on the back face; point; $\mathbf{v}_1$ is a boundary vertex of the facet, where the thickness is zero; $d_{\mathbf{n}}$ is the distance from $\mathbf{y}$ to the back face along the back-face normal $\mathbf{n}_b$; and $d_{\mathbf{y}}$ is the distance along the line of sight (shown as a dashed line). (a) shows the result when $\mathbf{v}_1$ is in front of $\mathbf{x}$. (b) shows the result when $\mathbf{x}$ is in front of $\mathbf{v}_1$.

The last equality uses the notation $\mathbf{y} = M_{F_f}\mathbf{w}_{F_f}(\mathbf{y})$, following similar convention as in (5).

Let $\mathbf{w}' = M_{F_b}^{-1}\mathbf{y}$. We know that $\mathbf{w}_{F_b}(\mathbf{x}) = s\mathbf{w}'$, and that $|\mathbf{w}_{F_b}(\mathbf{x})| = 1$. Therefore, $\mathbf{w}_{F_b}(\mathbf{x}) = \mathbf{w}'/|\mathbf{w}'|$, and $s = |\mathbf{w}|^{-1}$.

We now observe that M_{F_b} and M_{F_f} are constant per facet, and that $\mathbf{w}'$ is a linear transformation of $\mathbf{w}_{F_f}(\mathbf{y})$, which in turn contains the nontrivial elements of $\mathbf{u}_f$. Therefore, if we determine the values of $\mathbf{w}'$ at the vertices of the facet, we can interpolate it linearly in the same fashion as $\mathbf{u}_f$. We already know that all the coordinates but one vanish at the outer vertices of the facet. All that is left for our vertex processing algorithm is to find $\mathbf{u}'$, obtained from $\mathbf{w}'$ by adding the trivial barycentric coordinate $u_b = 0$, at the thick vertex and pass it as a second texture coordinate vertex attribute. The fragment program code only needs to normalize it.[1]

Thickness interpolation. From the above discussion, we can derive directly a simple expression for the thickness of the tetrahedron at the fragment, or the distance between front and back intersection points. Let us send $\mathbf{y}$, the eye-space position, as a vertex attribute that is interpolated for the fragment. We already obtained the interpolated $\mathbf{u}'$, and computed $s = |\mathbf{u}'|^{-1}$. Following this, we can find $\mathbf{x} = s\mathbf{y}$, and the thickness is

$$d_{\mathbf{y}} = \|\mathbf{x} - \mathbf{y}\| \tag{7}$$

Figure 3 illustrates a second method of interpolating the thickness, which does not depend on the barycentric coordinates, and is currently used in our program. It uses the position $\mathbf{y}$ and the unit-length normal $\mathbf{n}_b$ of the *back face*. For any set of parallel lines (such as the lines of sight in an orthographic projection), the distance between the front and back intersection points *is* interpolated linearly on the facet. Therefore, we can compute the distance $d_{\mathbf{n}}$ along $\mathbf{n}_b$ from the thick vertex $\mathbf{f}$ to F_b, send it as a vertex attribute, and linearly interpolate it between $\mathbf{f}$ and the outer vertices, where it is zero. The line-of-sight distance is now

$$d_{\mathbf{y}} = \frac{d_{\mathbf{n}}(\mathbf{y})}{\cos\theta} \tag{8}$$

where θ is the angle between the line of sight and $\mathbf{n}_b$. In practice, we don't need to compute θ directly, as we observe that $\cos\theta = \frac{-\mathbf{y}\cdot\mathbf{n}_b}{\|\mathbf{y}\|}$.

[1] Note for clarification that except for the outer edge of the facet, $\mathbf{u}' \neq \mathbf{u}_b$, and that normally $\mathbf{u}'$ does not hold true barycentric coordinates.

3.4 The final integral formula

The final line-integral formula, developed from (2), must include the coefficients of all the terms:

$$\int_0^1 f^j(\mathbf{u}(t))dt = \sum_{|\mathbf{k}|=d} \beta_{\mathbf{k}}^j \int_0^1 B_{\mathbf{k}}^d(\mathbf{u}(t))dt \tag{9}$$

Our implementation of the fragment program uses texture memory to store the coefficients. We send the index j of the current cell along with the facet data, and the fragment program uses it to determine the coordinates of the texels that contain the coefficients. We use 32-bit floating point numbers for the coefficients, and we can store up to four coefficients in each texel (using the color channels). The total number of coefficients for a barycentric Bernstein polynomial f of degree d with n variables is $n_f = \binom{n+d-1}{n-1}$. In our case, $n = 4$ and so $n_f = \binom{d+3}{3}$. For $d = 0,1,2,3$, we have $n_f = 1,4,10,20$ respectively. We map j to a consecutive set of texels, and the fragment program makes $n_t = \lceil \frac{n_f}{4} \rceil$ texture-read operations.

Polynomials of different degrees have different integral formulas, and require a different number of coefficients. To accomplish this, we generate the code of the fragment program on the fly, during the initialization of the application after the mesh data has been read. Our algorithm enumerates all the basis functions $\{B_{\mathbf{k}}^d\}$, and for each one expands the expression for the low-order term product in (4), then multiplying the sum of products by the corresponding coefficients, line length, and $(d+1)^{-1}$ (which is a hard-coded constant in the fragment program). The final fragment program evaluates the integral as an expression without any looping, branching, or conditional evaluation.

Advanced graphics hardware features. The input CT data is given in CT numbers, which are integers between 0 and 4095. The coefficients of the polynomials, which approximate these numbers, are on a similar scale. Therefore, we need more than 8 bits for each coefficient in the texture. We are currently using 32-bit floats as texture elements. In addition, the final output has a dynamic range on the order of 10^5, which again requires a high-dynamic-range frame buffer. Currently, we are using 16-bit floating point pbuffer as the rendering context. This does not capture the full dynamic range of the output, since a 16-bit float uses 5 bits for the exponent, which can encode values up to the order of 2^{16}. 32-bit pbuffers, on the other hand, do not support OpenGL blending operations. Blending is essential for fast accumulation of the integral values from different tetrahedra. We tried other methods for accumulating the results, such reading the old value at the pixel to the fragment program, adding the new integral and writing the result back to the pixel, but this slows down the program significantly.

To answer the dynamic range problem, we scale the model to a bounding box whose maximal edge length is 2 (from -1 to 1), and scale and biasing the coefficients to a range between 0 and 1, before the rendering begins. The effect of the geometry and coefficient scaling is a linearly proportionate scaling in the magnitude of the integrals. The effect of the coefficient bias is adding the bias multiplied by the thickness of *the mesh* along the line of sight to the final integral. This means that we have to accumulate the thickness $d_{\mathbf{y}}$ in addition to the integral values. We do this by outputting $d_{\mathbf{y}}$ through the alpha component.

After reading the pbuffer content, we re-adjust the numbers for the scale and bias that we had performed to get the actual integral values back. The scale and bias technique can also assist in displaying the image on a limited-dynamic-range monitor.

For fast data exchange between the main program and the GPU, we are using a vertex-buffer object (VBO) to write the vertex attribute to the GPU, and a pixel-buffer object to read back the results.

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

Table 1: Dynamic ranges in Figure 4.

3.5 Algorithm summary

To summarize this section, let us review the inputs and outputs of the fragment program.

Inputs

- The barycentric coordinates on the front face, $\mathbf{u}_f$, are interpolated linearly over the facet.
- The eye-space position on the front face, $\mathbf{y}$, is interpolated linearly over the facet.
- The barycentric coordinates on the back face, $\mathbf{u}_b$, are interpolated through $\mathbf{w}' = M_{F_b}^{-1}\mathbf{y}$, which in turn is interpolated linearly over the facet.
- The back-face normal, $\mathbf{n}_b$, is constant per facet.
- The thickness along the back-face normal, $d_{\mathbf{n}}$, is interpolated linearly for the facet.
- The cell index j is constant per facet (and per cell).
- The density coefficients for the cell, $\{\beta_{\mathbf{k}}^j\}$, are read from texture memory indexed as a function of j.

Outputs

- The final integral value $\int_0^1 f^j(\mathbf{u}(t))dt$.
- The Cartesian distance between the intersection points: $d_{\mathbf{y}} = \|\mathbf{x}-\mathbf{y}\|$.

4 Experiments and Results

4.1 Comparison with a voxelized dataset

Our mesh structure is generated from a segmented CT dataset, using an algorithm developed by Mohamed [8]. The density functions are obtained by fitting polynomials to the CT intensity numbers in the space of each cell. Naturally, one wishes to compare images rendered from our mesh with images rendered from the voxelized CT dataset. However, a meaningful numerical comparison of the methods is difficult to achieve. The reasons are, first, that the topology and geometry of the two data sources are different. Specifically, the shape of the boundary of the object is different when the object is discretized as a tetrahedral mesh and as a rectilinear grid. Therefore, we should expect larger-scale differences near the object boundary when we compare the two representations. Second, the quality of approximation in the mesh representation of the CT study depends highly on the resolution (cell size) of the mesh and on the complexity (i.e., degree) of our density functions. Before comparing DRR results, we should ask how close our model is to the "ground-truth" CT, and this is beyond the scope of this paper.

In spite of these issues, we can provide a qualitative idea of the similarity between images of our mesh and DRRs projected from a CT study. To do this, we first use the geometry of the mesh as a binary mask to filter out all the CT voxels that are not covered by the mesh. This approximates the object shape as equally as we can in the two representations. We create DRRs of the segmented CT using the Take package [10], and images of the mesh in equivalent camera poses. We can then compare these image sets.

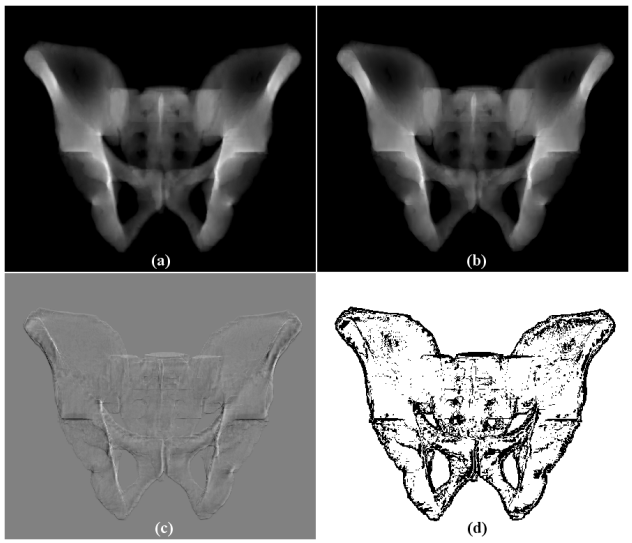

Figure 4: Comparing DRRs from a segmented pelvis CT and a tetrahedral mesh. (a) Anterior-posterior (AP) view of CT DRR. (b) AP view of mesh DRR with 3^{rd} order density functions. (c) Difference image (b)-(a). (d) Locations of relative difference $|E(\mathbf{p})| \geq 0.05$.

Figure 4 shows an example of such a comparison. We used a model of the pelvis bone, with 14,375 vertices and 52,837 tetrahedra, and 3^{rd} order density polynomials, shown as (b), and compare it with the segmentation of the CT study that produced the mesh (a). The visual differences between the two DRRs are small. A difference image (c) shows that the main differences lie near the object boundaries. Details about the dynamic range of these images are summarized in Table 1.

We are using a pixelwise relative error measure, defined as $E(\mathbf{p}) = \frac{I_{Model}(\mathbf{p}) - I_{CT}(\mathbf{p})}{I_{Model}(\mathbf{p})+1}$ (we add 1 at the denominator to prevent division by zero). Image (d) shows as black pixels the locations where $|E(\mathbf{p})| \geq 0.05$. As we can see, the relative error in most of the image area is less than 5%.

4.2 Performance analysis and degree comparisons

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 twice over a camera trajectory of 25 frames in 9° increments (a 216° arc). The image size was 512^2 pixels. The performance results are summarized in Table 2. All the timing experiments were performed on a dual Xeon PC, 2.80GHz CPU speed, with NVidia GeForce 6800 GT graphics card, and under Windows XP.

To understand the contribution of using higher-order polynomials to the DRR, we subtracted images of the mesh with degree $d-1$ from images of degree $d = 1,2,3,4$. An example of the difference images for the anterior-posterior (AP) view is in Figure 5. For a numerical comparison, we summarize the dynamic range of the 25 projections, the dynamic range of the difference $I_d - I_{d-1}$, and the root of mean square difference (RMS) for all 25 projections as a single of pixels in Table 3. 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 over-fitting.

Degree	0	1	2	3	4
Tessellation and vertex attributes (CPU): ms/frame	283.4	284.7	283.4	283.6	284.0
OpenGL rendering (vertex & fragment program): ms/frame	29.7	30.4	61.4	148.5	1049.4
Number of instructions in compiled fragment program ($n_i(d)$)	31	45	102	276	750
Number of terms and coefficients (n_f)	1	4	10	20	35
Number of texture-read operations in fragment program	1	1	3	5	9

Table 2: Comparing the performance of rendering 50 frames of the pelvis mesh, with different polynomial degrees.

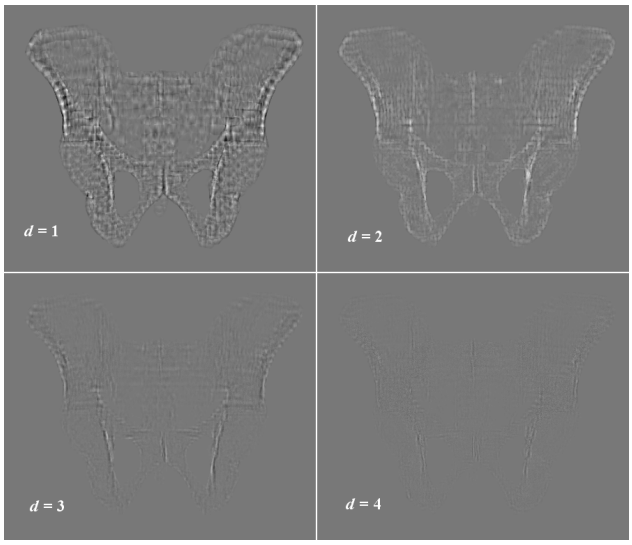

Figure 5: Difference between images of the pelvis mesh with polynomials of degree d and degree $d-1$. The images show *signed differences*, with negative values in black, zeros in gray, and positive differences in white. The values of the differences are scaled equally in all four images. d goes from 1 to 4.

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

Table 3: Numerical values in DRR images of pelvis mesh with different degrees. d is the degree of the density functions. The table shows the maximum and minimum intensity values over a trajectory of 25 images, and 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.

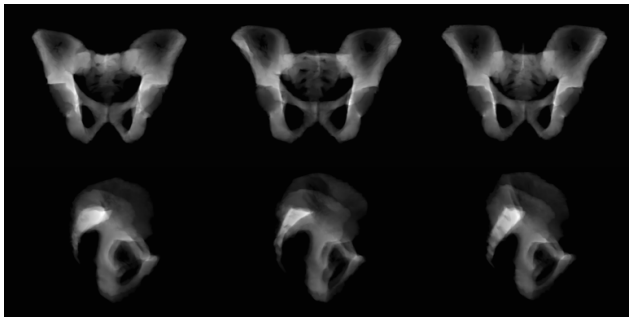

Figure 6: Examples of shape variations in a pelvis model. The mean shape of a 6-subject population is shown on the left. Some typical shape variations are in the middle and on the right. The top row shows AP projections, and the bottom shows lateral projections of the same shape, that is, the model is rotated 90° around the vertical axis.

4.3 Mesh deformations

The work presented in this paper is part of a project of creating a statistical atlas of bone anatomy. The atlas contains information about the typical (average) shape of a specific anatomy, and about typical variations in the shape. An important component in the statistical atlas project is the ability to use it in a 2D-3D registration framework, that is, match an observed set of 2D patient x-ray images with a pose and a shape of the statistical model. For this, we need an efficient tool for creating images of the deformed atlas, and comparing them with the patient images.

We created a statistical atlas of the pelvis bone, which consists of a "master mesh" $\mathbf{M}_0$, created from a "master dataset" CT_0. CT_0 is elastically registered to instance datasets $\{CT_i\}$ [12], and an equivalent deformation is applied to the vertices of $\mathbf{M}_0$ to generate instance meshes $\{\mathbf{M}_i\}$. The final atlas is determined by the average vertex positions in the instance meshes, and by extracting deformation modes using PCA (cf. [1]). In addition, we refitted the density polynomials for each instance dataset, and computed the average density function for the atlas. This process generally follows the atlas creation process described by Yao in [18].

The deformation modes resulting from the PCA are used as displacements that are applied in a linear combination to the vertices in the mean shape. If $\mathbf{v}_j$ is the position of the j-th vertex in the mean shape, the deformed position is

$$\mathbf{p}_j(\bar{\gamma}) = \mathbf{v}_j + \sum_{k=1}^{n_m} \gamma_k \mathbf{m}_{kj}$$

where n_m is the number of deformation modes; $\mathbf{m}_{kj}$ is the displacement for the vertex $\mathbf{v}_j$ given by the k-th mode; and $\bar{\gamma} = (\gamma_1, \ldots, \gamma_{n_m})$ are scalar weights of the deformation.

Our population currently consists of 6 subject datasets. The mesh includes 12,203 vertices and 44,151 tetrahedra, with 3^{rd} order density functions. The mean shape and some typical variations are shown in Figure 6.

The attached video shows an interactive interface for deforming the mesh. The user can select which deformation mode to activate, then drag the mouse up and down to apply a different magnitude of this mode. The atlas in the video includes two deformation modes.

4.4 Usage in 2D-3D deformable registration

We have implemented a framework for 2D-3D registration between static target images and dynamic DRRs created from the atlas

(model images). The registration algorithm searches for the maximum of a *mutual information* (MI) similarity measure between target and model images. The transformation parameters include translation, rotation, and shape deformation, which is implemented as a linear combination of displacements, as described above.

We ran a series of control experiments, with synthetic deformation modes and known transformations. In these, our registration framework recovered the parameters to a precision of about 0.01 millimeters in the translation, 0.01° in the orientation, and 0.01 in the values of the displacement coefficients. In one example, we used two orthogonal views of size 256^2, and a Downhill Simplex optimizer (taken from the `vnl` library [4]). In each iteration, the optimizer generates two DRRs (which is the number of target images), reads back the data, and computes the MI similarity. The number of iterations was 328, and total time for registration was 233.219 seconds, or 355.5 ms/frame (with two frames rendered per iteration).

In another registration experiment, the atlas was created from a five-subject population, and the target images were generated from a sixth subject dataset, not included in the atlas. Our framework recovered a good approximation of the pose of the target dataset, but a less satisfying shape matching. A five-subject database is likely too small to cover the possible variability in human anatomy. Nevertheless, our 2D-3D registration framework, which uses the DRR algorithm seems to recover pose and shape to the best extent covered by the training data. Results of these studies will be published separately.

An example of the registration progress is shown in the attached video. The static target image is on the left in cyan, and the dynamic deformable model image is in red, shown overlaid on the target image and isolated on the right. The optimization first searches for a translation, then for a rotation, and then for a shape deformation. Then this sequence is repeated for fine tuning. A final part searches through all the parameters. A good registration is indicated by blending equal intensities of red and cyan into a gray result. The image on the right is shown for better tracking of the changes in the pose and shape of the model. It can be observed that after the first round of optimizations, we already have a fairly good registration, and the improvements of the second round are barely visible (see the acetabulum on the right side, and the top of the sacrum). Note that the video shows a time-compressed process, by a factor of about 15.

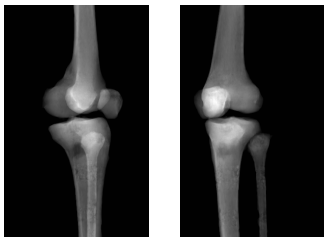

Figure 7: DRR images of a tetrahedral model of bones segmented from a knee phantom CT scan, including: femur, patella, tibia, and fibula. Two views are shown. The images were created using our algorithm.

5 Discussion

5.1 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 Tables 2 and 3, 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.
- The size of the integral expression (9), 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 computation time on the GPU increases even more rapidly than the size of the fragment program. Note that this time includes access to texture memory, and the number of texture operations seems to have a significant impact on the performance. Between $d = 0$ and $d = 1$, there is hardly an increase in the GPU time, although the size of the program increases about 1.5 times. We believe this is affected by the single texture access, compared with the cubically increasing number of texture accesses required in higher degrees. An increased number of texture accesses creates potential for cache misses and for blocking between concurrent fragment programs trying to access the limited texture resource, both of which can slow down the process significantly.
- The relative contribution of increasing the degree of the polynomial to the detail of the image diminishes, as can be seen in Figure 5 and Table 3.

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 Equation (4). This requires a more sophisticated generator of the fragment program source code, and a better understanding of the structure of the formula.
- Try other methods of passing the density coefficients, for example by using shared registers (Cg keyword `uniform`). Although loading the registers for every cell may take some time, this time may be saved when it comes to reading texture memory. Another option is to pass the coefficients through the vertex attributes, but then the number of coefficients and the degree of the polynomial are bounded by the number of available attribute registers, more memory is required per vertex, and more data needs to be sent to the rendering pipeline. We can also reduce the number of texels required for the coefficients by using fewer bits per coefficient. This is likely to reduce the problem of many texture reads, but is still a limited solution.
- Use more efficient per-vertex parameter passing, avoiding replications such as the back-face normal. It appears that these should have a minor effect compared to optimizations in the integral expression and texture access.
- Encode the tessellation algorithm in a vertex program, following, for example, [17]. If done properly, this can contribute to the speedup of the algorithm, but eventually the performance bottleneck of the integration should override the tessellation time.
- Balance the computational load between CPU and GPU. Currently, the CPU tessellation and the GPU rendering are done serially, but they may be done concurrently by proper adjustment and synchronization of the algorithm steps. With the current implementation of the integral formula, we can still

parallelize these parts up to 3^{rd} order functions without an increase in processing time.

Our results suggest that in terms of precision, 3^{rd} 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 level, which can be expressed as the difference between the intensity values in a CT study and the values predicted by 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.

5.2 Deformations, intensity variations, and registration

In the current implementation of our algorithm, applying a deformation to the mesh takes practically no toll on the performance, as can be seen in the results. However, if we choose to implement the tessellation algorithm as a vertex program, we should adopt a new strategy in applying the deformation. For example, we may store all the deformation modes on GPU buffers, and have a two-pass vertex processing method: The first pass computes a linear combination and outputs deformed vertex positions. The second pass performs the tessellation.

Applying variations in the density properties of the mesh, requires reloading the texture memory with the changed coefficients. This may slow down the performance, and needs to be tested. A possible optimization is to store the density variation modes as separate textures, and apply them by running a fragment program that takes weights as inputs, and outputs the linear combination of the variations using these weights.

We have a working 2D-3D registration framework, yet the optimization algorithm, Downhill Simplex, seems to be less efficient than what we desire. We have examined several gradient-based methods, with no better answer yet, and we will continue looking at this problem.

6 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. The algorithmic focus of this paper is on obtaining the correct values of the parameters used in the integral formula through interpolation and texture lookup.

The experimental results presented here show that using higher-order attenuation functions contributes to the quality of the final image, 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, and the multiple texture access operations, required for fetching the function coefficients, seem to impact the performance quite badly. These problems should be addressed in future research.

In spite of these issues, we believe that the other contributions of this work outweigh the drawbacks. We present elegant geometric solutions for interpolation of thickness and barycentric coordinates. Even though we haven't fully optimized the GPU programs, their performance is at least 10 times faster than the older method we used, which rasterized the tetrahedra and computed the integrals on the CPU. The ability to include shape and density variations in the rendered image is a key element in studying and using generic anatomical model, created from a population study. We consider this a leap forward in the application environment of computer assisted surgery.

7 Acknowledgments

We would like to thank the following people for their help in creating the statistical pelvis atlas: Gouthami Chintalapani, Lotta Ellingsen, Ashraf Mohamed, Anna (Na) Song. The pelvis CT datasets were given to us by Drs. Ted DeWeese and Lee Myers. This work was supported in part by NSF ERC Grant EEC9731478, by NIH/NIBIB research grant R21-EB003616, and by NSF ITR Grant AST-0313031.

References

[1] T F Cootes, C J Taylor, D H Cooper, and J Graham. Active shape models – their training and application. *Computer Vision and Image Understanding*, 6(1):38–59, 1995.

[2] R Farias, J Mitchell, and C Silva. Zsweep: An efficient and exact projection algorithm for unstructured volume rendering. In *Proceedings of 2000 ACM/IEEE Volume Visualization and Graphics Symposium*, pages 91–99, 2000.

[3] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM Press.

[4] The VXL Group. The vxl library. http://vxl.sourceforge.net/.

[5] Martin Kraus, Wei Qiao, and David S. Ebert. Projecting tetrahedra without rendering artifacts. In Holly Rushmeier, Greg Turk, and Jarke J. van Wijk, editors, *IEEE Visualization 2004*, pages 27–33, October 2004.

[6] David A. LaRose. *Iterative X-ray/CT registration Using Accelerated Volume Rendering*. PhD thesis, Carnegie Mellon University, 2001.

[7] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, New York, NY, USA, 1996. ACM Press.

[8] Ashraf Mohamed and Christos Davatzikos. An approach to 3d finite element mesh generation from segmented medical images. In *IEEE International Symposium on Biomedical Imaging (ISBI)*, 2004.

[9] Kenneth Moreland and Edward Angel. A fast high accuracy volume renderer for unstructured data. In *IEEE Symposium on Volume Visualization and Graphics 2004*, pages 9–16, October 2004.

[10] Jens Muller-Merbach. Simulation of x-ray projections for experimental 3d tomography. Technical report, Image Processing Laboratory Department of Electrical Engineering Linkoping University, SE-581 83, Sweden, 1996.

[11] Stefan Röttger, Martin Kraus, and Thomas Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings of IEEE Visualization 2000*, 2000.

[12] D Shen and C Davatzikos. Hammer: Hierarchical attribute matching mechanism for elastic registration. *IEEE Trans. On Medical Imaging*, 21(11):1421–1439, November 2002.

[13] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. In *Proceedings of the 1990 workshop on Volume visualization*. ACM Press, New York, NY, USA, 1990.

[14] Cláudio T. Silva, Joseph S.B. Mitchell, and Peter L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *IEEE Symposium on Volume Visualization*, pages 87–94, 1998.

[15] Manfred Weiler, Martin Kraus, and Thomas Ertl. Hardware-based view-independent cell projection. In *IEEE Volume Visualization and Graphics Symposium 2002*, pages 13–22, 2002.

[16] Peter L. Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2), April 1992.

[17] Brian Wylie, Kenneth Moreland, Lee Ann Fisk, and Patricia Crossno. Tetrahedral projection using vertex shaders. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*. IEEE Press, Piscataway, NJ, USA, 2002.

[18] Jianhua Yao. *A statistical bone density atlas and deformable medical image registration*. PhD thesis, Johns Hopkins University, 2002.