Interactive Volume Isosurface Rendering Using BT Volumes

John Kloetzli
UMBC

Marc Olano
UMBC

Penny Rheingans *
UMBC

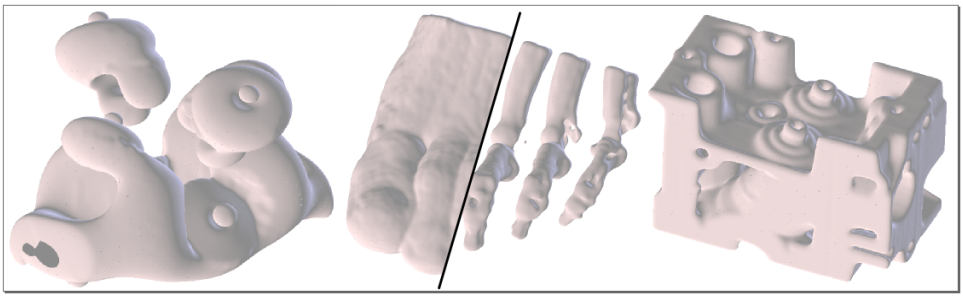

Figure 1: *Several renderings of BT Volume data. Left: Real-time isosurface rendering of a molecular simulation. Center: Composition of two real-time isosurface renderings of a foot data set. Our system renders any isosurface level and supports changing the isosurface level in real time. Right: Higher detail rendering of engine block generated offline.*

Abstract

This paper presents a volume representation format called BT Volumes, along with a technique to interactively render them and two methods to create useful data in BT Volume format, including high quality reconstruction filtering. Medical applications rely heavily on isosurface data to visualize anatomy, but current real-time isosurface rendering techniques such as Marching Cubes are limited in flexibility and provide only low-order linear reconstruction filtering. As an alternative to creating triangular geometry to represent the surface, we ray trace an exact isosurface directly inside a pixel shader. We construct a set of Bézier Tetrahedra to approximate any reconstruction filter with arbitrary footprint. We then precompute the volume convolved with this filter as a tetrahedral grid with Bézier weights that can be ray traced in graphics hardware. Our technique is fast, renders any isosurface level without additional work, and performs high quality reconstruction filtering with arbitrary footprints and reconstruction kernels.

Keywords: Volume Rendering, Isosurface Rendering, Ray Tracing, Graphics Hardware, Bézier Tetrahedra, Real Time, Interactive, BT Volumes

1 Introduction

Volume rendering converts volumetric data into meaningful 2D images. Applications of volume rendering range from hurricane visualization to medical diagnosis and planning to smoke and particle systems. There are many types of volume rendering which vary greatly in applicability and style. We focus specifically on *isosurface rendering*, where we display a surface representing the locus of points in the volume matching a user-specified value. They form a 2D surface in 3D space similar to the way contour lines on a topographical map form 1D lines on a 2D map. Isosurface rendering is particularly popular in medical visualization, where it effectively addresses a number of unique challenges. Many medical scanning techniques return volumetric scalar grids representing physical quantities. For example, Computed Tomography (CT) scans measure density, while Magnetic Resonance Imaging (MRI) scans measure the resonant response of hydrogen atoms in the tissue to specific frequency RF pulses. Areas of interest in medical volumes often coincide with specific value boundaries. For example, a doctor seeking to localize a tumor for radiation treatment planning is interested in the exact boundary of the tumor, while a doctor working on a complex cranio-facial reconstruction in interested in the precise shape of the skull.

We propose a new method for representing volumetric data which allows rendering arbitrary isosurfaces of a high-quality reconstruction of a volume. Our method works by approximating an arbitrary reconstruction filter with a set of cubic Bézier Tetrahedra. We show that convolution of this reconstruction filter with a volume is equivalent to 'collapsing' the reconstruction filter into a tetrahedral mesh of the volume. We call this representation of the volume as Bézier Tetrahedra a BT Volume. We can interactively render the isosurface within each tetrahedron using *local ray tracing* within the tetrahedral volume. Our choice of cubic Bézier Tetrahedra as a basis allows the exact ray intersection equations to be computed within each tetrahedron on the GPU.

Isosurface renderings of volumes generally have large empty spaces where the volume density was either well above or below the isosurface value. Most isosurface rendering methods create triangle meshes for the isosurface of interest and display that mesh using established 3D polygonal rendering techniques. This allows them to extract the isosurface information while ignoring areas outside of the range of the current isosurface value. The most common of these belong to the *Marching Cubes* family of techniques [Lorensen and Cline 1987]. In contrast, our method does not create a triangle mesh, but renders isosurfaces directly using ray tracing.

Marching Cubes methods are based on a *linear reconstruction* of the volume. Since each triangle is planar, the resulting surface will always have a faceted appearance caused by the flat triangles. Linear reconstruction is considered a poor method for creating continuous functions from a discrete sampling. Higher quality reconstruction techniques also exist but are much more expensive to evalu-

*e-mail: jk3,olano,rheingan@cs.umbc.edu

ate. Cubic reconstruction of a volume is considered a high quality method but requires evaluating a 64-component summation as opposed to the eight values required for the linear technique. High quality rendering of volumes is not attempted in real time because of the processing power and data bandwidth required. Our technique changes this by representing the high quality reconstruction of a volume (including cubic and higher order filters) in a format which can be rendered quickly.

Recently, consumer-level Graphics Processing Units (GPUs) have reached a height of performance which far surpasses current CPU technology. Of particular interest are the variety of new *local ray-tracing* techniques, which use the traditional graphics hardware pipeline to render simple primitives which serve as bounding boxes on the true surface being rendered. For each point on the bounding surface, the GPU pixel-level processors compute viewing ray-object intersection locally inside each bounding box [Policarpo et al. 2005; Tatarchuk 2006; Ritsche 2006; Loop and Blinn 2006]. This technique forms the basis of our method, which creates a regular grid of polynomial surfaces, called Bézier Tetrahedra, to represent the reconstructed surface and uses each tetrahedron as a local bounding box for ray tracing of the surface within. Our implementation makes efficient use of graphics hardware geometry and pixel shaders to render a smooth resolution-independent isosurface at interactive rates.

2 Previous Work

Discrete sampling and reconstruction in volumes has been under investigation for many years. Mitchell and Netravali [1988] present a thorough overview of the topic and a set of criteria for designing new high quality filters. In addition, they present a family of new cubic reconstruction filters which include Bézier and Catmull-Rom polynomials as special cases.

In order to deal with sampling and reconstruction within the domain of volume rendering, Marschner and Lobb [1994] performed an evaluation of common reconstruction filters in the context of volume reconstruction. They looked at linear, cubic, truncated Gaussian, cosine bell, and windowed sync filters, evaluating each one according to an error metric which included measurements of smoothness and several types of aliasing.

One common method for rendering isosurfaces of volumes is called *Marching Cubes*. It works by computing a triangle mesh for a specific isosurface value, using graphics hardware to render the triangle mesh [Lorensen and Cline 1987]. Since GPUs are designed to render triangles very quickly, these methods are fast once the triangle mesh has been created. Unfortunately, they necessarily use linear reconstruction to create the triangle mesh, which is generally considered a poor reconstruction filter. Even though this method has been around for more than 20 years, it is still very popular because it is easy to use, portable, and fast.

The basic Marching Cubes algorithm looks at eight adjacent sample points in the volume. Because of the linear reconstruction used, these sample points must surround the target isosurface value for the isosurface to have passed through this cube. Otherwise the cube is entirely inside or entirely outside the isosurface being rendered, and no work needs to be done. When this method is performed for all sets of adjacent sample points defining a cube, the entire isosurface can be roughly reconstructed. Many other methods based upon this technique have been developed, including methods which use different primitive shapes (*Marching Diamonds* [Anderson et al. 2005] and *Marching Tetrahedra* [Treece et al. 1999]) and real-time versions which take advantage of graphics hardware to accelerate the process [Johansson and Carr 2006].

An interesting extension to the Marching Cubes method by Theisel [2002] notes that the particular reconstruction method that Marching Cubes uses does not match standard linear interpolation. In fact, the contours along the side of each cube under linear reconstruction are curves, while in the isosurface resulting from Marching Cubes they will be lines. Theisel presented a modification for Marching Cubes that generates rational Bézier patches that exactly match the linear reconstruction instead of triangles.

Another class of fast isosurface volume rendering, which we call *direct volume* methods because they do not require preprocessing, works by directly intersecting each viewing ray with the isosurface being rendered. This typically involves solving ray-patch intersection equations, which are slow and cumbersome to evaluate in graphics hardware. Levoy [1990] developed a graphics hardware accelerated method to do this, while Parker et al. [1999] presented a brute-force version running on a cluster. A more advanced version of this approach by Rössl et al. [2003] found the optimal weights for quadratic BT on a simplical grid, ray tracing the resulting surface. Their method did not use graphics hardware acceleration.

3 Background

In order to describe our volume rendering method we first must describe some of the background knowledge our method is built upon. We discuss the applicable mathematics behind sampling and reconstruction of volumes, followed by the mathematical description of our rendering primitive, the Bèzier Tetrahedron (BT). Finally, we describe how BT can be rendered in real-time on current generation graphics hardware.

3.1 Sampling and Reconstruction

For the purposes of this paper we will discuss only regularly sampled scalar-valued rectilinear fields. Each point of data in these volumes is called a *sample point* and, in the absence of some reconstruction filter, the field is undefined between sample points.

Reconstruction is the process of creating a continuous function from a discrete volume. Reconstruction requires blending sample points from the volume using a filter *kernel* to define how the sample points should be blended together. In most signal processing contexts, a sinc filter is considered to provide perfect reconstruction, since it reproduces the maximum frequencies captured by the sampling while avoiding adding any higher frequencies. In image processing, the infinite radius of this filter and the ringing it generates make it less desirable than other filters which avoid these problems, though they may introduce more blurring or aliasing. Given this, we would like to be able to support a variety of reconstruction kernels of differing sizes.

The mathematical tool which we use to perform reconstruction filtering is called *discrete convolution*, which, in the context of volumes, is a function of 3D space that sums a reversed *kernel filter* with a volume. For some cuboid domain $C \in \mathbb{Z}^3$, the formula for the convolution of $A : C \to \mathbb{R}$ by a kernel $G : \mathbb{R}^3 \to \mathbb{R}$ at the point $\mathbf{P}$ in $R \in \mathbb{R}^3$ is the summation over all possible values of integer-valued (a, b, c) in the kernel domain given by

$$(A * G)(\mathbf{P}) = \sum_{a,b,c \in R} A(a,b,c) \cdot G(\mathbf{P} - (a,b,c)) \quad (1)$$

We will use this definition later to convolve our reconstruction filter with volume data.

3.2 Bézier Tetrahedra

Loop and Blinn [2006] developed a rendering method for a specific type of Bézier solid called a *Bézier Tetrahedra* which they implemented on graphics hardware. The BT are a set of polynomial solids in the Bernstein basis where each element of the family is defined by a bounding tetrahedra and a set of weights. The cubic BT formula, which is the order BT used in our method, can be written in terms of the points which define the bounding tetrahedra $\mathbf{T} = (\mathbf{T_1}, \mathbf{T_2}, \mathbf{T_3}, \mathbf{T_4})^T$ and the set of 20 weights $\{w_{ijkl} : i+j+k+l=3\}$. Given a point $\mathbf{P} = (x, y, z, 1) \in P\left(\mathbb{R}^3\right)$(three-dimensional Euclidian projective space), define the *barycentric coordinates* of the point to be $\mathbf{r} = (r, s, t, u) = \mathbf{P} \cdot (\mathbf{T})^{-1}$. The formula for the BT associated with the tetrahedron defined by the points $\mathbf{T}$ is given by

$$bt(\mathbf{P}) = \sum_{i+j+k+l=3} w_{ijkl} \binom{3}{ijkl} r^i s^j t^k u^l \tag{2}$$

3.3 Bézier Tetrahedra as Tensors

BT have a tensor form which allows transformation into different spaces. All tensors in this section are written in Einstein Index Notation. To write a BT in tensor form, consider the BT defined by a tetrahedron $\mathbf{T}$ and a set of weights $\{w_{ijkl} : i+j+k+l=3\}$. Construct a 4^3 tensor of control points $\mathbf{B}$ by

$$\mathbf{B}_{\alpha_1,\alpha_2,\alpha_3} = \mathbf{w}_{\mathbf{e}_{\alpha_1}+\mathbf{e}_{\alpha_2}+\mathbf{e}_{\alpha_3}+\mathbf{e}_{\alpha_4}}$$

where e_α is a four component vector with a 1 at position α and all other components equal to 0. This formulation of a BT weight set is much less compact than the original formulation from equation 2 (64 scalar values compared to 20 in the original version), so our implementation never stores the tensor form, instead expanding from the tetrahedral form to compute transformations and encoding the result back into the tetrahedral form.

One additional useful feature of Bézier Tetrahedra is the *bounding property*, which describes limits on the range of the solid. For a given BT with weights $w_{i,j,k,l}$, any value resulting from evaluation of the BT will have to be between the highest and lowest weight values. In other words, for any point $\mathbf{P}$ in the domain of a BT with weights $w_{i,j,k,l}$ the resulting value $Q = bt(\mathbf{P})$ must be between the maximum and minimum values in the weight set $w_{i,j,k,l}$. Therefore, one can determine easily if a given BT has an isosurface of a specific level by computing the min and max weights. We use this fact to perform early culling of tetrahedra which cannot contain any of a particular isosurface level.

3.4 Ray Tracing Bézier Tetrahedra

Our BT ray-tracing method is an extension of the method first described by Loop and Blinn [2006], described here. The process of developing a fast ray-tracing method for BT involves several steps, including transformation of the weight tensor and bounding tetrahedra into screen space and solving the ray-BT intersection equations.

The transformation into screen space is defined by a 4x4 transformation matrix called the World-View-Projection matrix, denoted $\mathbf{WVP}$. Because the vertices of the bounding tetrahedron $\mathbf{T}$ are in Euclidean space already, applying the World-View-Projection transformation directly will transform them into screen space. The BT weights, however, are in the barycentric space defined for equation 2 above, so we will need to multiply the transformation from barycentric coordinates into Euclidean space ($\mathbf{T}$) with the World-View-Projection transformation. The inverse of this composite transformation provides us with an equation for the barycentric coordinates $\mathbf{r}$ of a screen space point $\mathbf{P_s}$ given by

$$\mathbf{r} = \mathbf{P_s} \cdot (\mathbf{T} \cdot \mathbf{WVP})^{-1} = \mathbf{P_s} \cdot \mathbf{W}$$

where W is the inverse composite transform. This makes the transformed weight tensor

$$\tilde{\mathbf{B}}_{\beta_1,\beta_2,\beta_3} = \mathbf{W}^{\alpha_1}_{\beta_1} \mathbf{W}^{\alpha_2}_{\beta_2} \mathbf{W}^{\alpha_3}_{\beta_3} \mathbf{B}_{\alpha_1,\alpha_2,\alpha_3} \tag{3}$$

The zero-isosurface of the transformed BT is

$$\mathbf{P_s}^{\beta_1} \mathbf{P_s}^{\beta_2} \mathbf{P_s}^{\beta_3} \tilde{\mathbf{B}}_{\beta_1,\beta_2,\beta_3} = 0 \tag{4}$$

Finally, we must evaluate the intersection of equation 4 with each viewing ray. We refer the reader to the work by Loop and Blinn [2006] for a more detailed explanation of this process. The fundamental observation is that in screen space each viewing ray has constant x and y values, only varying in the z direction. Therefore, plugging the equation for a viewing ray into equation 4 will collapse the left side of the equation from a polynomial with four variables into a univariate polynomial in z. The roots of that equation define the intersection points of the ray with the surface.

4 Method

We present a method for representing volumetric functions as a set of Bézier Tetrahedra and describe how to render these volumes efficiently on graphics hardware. This involves dividing a cuboid volume into a tetrahedral grid following certain regularity properties and using BT to define density within each grid point. We use this representation, called BT Volumes, to store a volumetric function in a form where any isosurface can be rendered quickly. With that alone, we have a volumetric extension of the work of Loop and Blinn [2006], but like their work, authoring of these BT datasets could be a problem.

Section 4.2 presents two ways to generate useful BT Volumes. The first method shows how to approximate any continuous scalar volumetric function with a BT Volume using a least-squares technique. This approximation can be arbitrarily close depending on how many tetrahedra comprise each voxel of the BT Volume.

The second way uses an existing BT Volume as a reconstruction kernel which can be convolved with any discrete volume to generate a new BT Volume. This means that we can exactly render isosurfaces of the reconstruction of any scalar volume with a BT Volume filter, which allows us to generate a high-quality, smooth, resolution-independent representation of a convolved volume.

4.1 Representing Volumetric Functions

There are several steps to storing volumetric data as a BT Volume. First, we divide the volume space (which, without loss of generality, we assume to be a cuboid with integer dimensions) into unit cubes. We refer to each of the cubes as "voxels" because our reconstruction technique maps each voxel of the input volume into one of the cubes. We then define the "canonical mapping" to create a way of dividing each voxel into tetrahedra. Our implementation uses the standard 6-tetrahedron method of breaking a cube into tetrahedra, but many other methods can also be used as described in Section 4.1.3. This results in the entire volume space covered by a grid of tetrahedra such that every point intersects at least one tetrahedron.

Now that our tetrahedral grid has been created, we associate a BT with each of the grid points and develop a transformation pipeline which allows us to render any isosurface of these BT. Since a BT

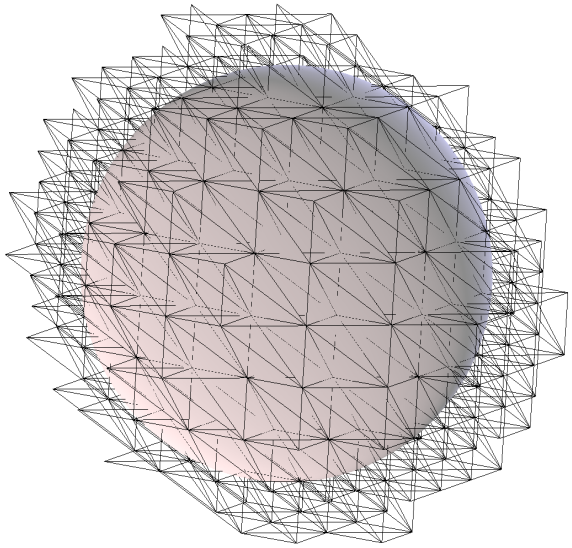

Figure 2: *An isosurface rendering of a BT Volume approximating a Gaussian reconstruction filter where lines denote the boundaries between adjacent tetrahedra. Note that tetrahedra which would not contribute to this particular isosurface are culled before rendering. We generate filters which are C^0 continuous and enforce zero values on the boundary.*

is defined inside its bounding tetrahedron, each point in the volume now has a density value defined by the BT associated with that particular tetrahedral grid point. With correctly defined weights this representation allows many continuous volumetric functions to be represented. A little extra work (beyond that described by Loop and Blinn [2006]) must be done to allow ray tracing of arbitrary isosurfaces, since the ray-tracing method we presented earlier assumed that we were generating the zero-isosurface of the solid.

Note that, like Loop and Blinn's original BT-rendering paper [2006], this section assumes a BT Volume already exists. We are inherently limited by the expressiveness of the Bézier Tetrahedra primitive which we use — not every volumetric function has an exact BT Volume representation — and by our ability to author BT Volumes. To handle arbitrary volumes, we need the filtering and fitting tools described in Section 4.2.

4.1.1 Foundations

The first step in defining BT Volumes is to divide the volume domain into 3D cubes, breaking each cube into tetrahedra. Each tetrahedron is associated with a BT solid to represent the portion of the volume within the tetrahedron. This division into BT ensures that every position in the volume is "covered" by at least one BT — positions exactly on the boundary between tetrahedra can be considered as covered by either BT.

Notationally we will refer to each of the small cubes which comprise the volume space by $c_{l,m,n}$, where l, m, and n denote the (x, y, z) position within the volume space. We will call each of these cubes "voxels" of the BT Volume because they are analogous to voxels of standard volumes. In fact, our convolution method described later will generate one BT Volume voxel for every voxel in the original volume data. For convenience, we assume that BT Volume voxels, as well as standard volume voxels, are unit-sized.

4.1.2 Abstract Mapping for Tetrahedra

We will divide a BT Volume voxel into tetrahedra by the following function, called the *canonical mapping*.

$$\kappa : [0, 0, 0] \times [1, 1, 1] \rightarrow \{h : h \in H\} \tag{5}$$

where H is the set of unique indices for each tetrahedron defined by $\{h : h \in \mathbb{Z}, [1...n]\}$ where n is the number of tetrahedra comprising each voxel. The important feature of κ is that, because it maps from the unit cube, each cube must divide into tetrahedra in the same way and therefore the entire tetrahedral partition is shift invariant. Because our rendering method does not rely on a specific κ, we present an abstract definition which provides a minimum set of properties required. Let us define $\bar{BT}_{lmn}$ to be the set of all BT in voxel $c_{l,m,n}$, where each member has a *tetrahedron number*, $h \in H$, which acts as a unique identifier within that voxel. Let the mapping

$$BT_{lmn}(h) = bt^h \in \bar{BT}_{lmn} \tag{6}$$

denote the BT (bt^h) associated with index h. The BT in the volume corresponding to a point $\mathbf{Q} \in \mathbb{R}^3$ is then denoted by

$$\tau = BT_{\lfloor \mathbf{Q} \rfloor}(\kappa(\mathbf{Q} - \lfloor \mathbf{Q} \rfloor)) \tag{7}$$

4.1.3 Specific Mappings for Tetrahedra

One simple κ is to create six BT for each voxel c_{lmn}, packing the voxel using the standard method for dividing a cube into six tetrahedra. This is the formula we have used in our implementation, but other mappings are possible and there are several properties to consider when deciding which to use [Carr et al. 2006]. The top of Figure 2 pictures a BT approximation of a Gaussian function using this method, which generates 1296 BT for a volume with 6^3 voxels.

4.1.4 Representing any Isosurface Level

We have introduced Bézier Tetrahedra in equation 2 as a polynomial solid which resides inside the barycentric coordinates defined by a bounding tetrahedron, $\mathbf{T}$. In Section 3.4 we extended our understanding of BT by showing 1) how we can represent them in *tensor* notation and 2) how this new tensor formulation can be transformed into spaces other than the barycentric coordinates used in the definition. In this section we use these two facts to develop a new way of rendering arbitrary isosurfaces of a BT.

The final BT isosurface equation we presented in equation 4 for ray tracing represents the zero-density isosurface of the solid. However, we want to be able to render arbitrary isosurfaces in our volume. Fortunately, adding a constant value to a BT results in another BT, so that for any BT a there exists another BT b such that the x-isosurface of a equals the zero-isosurface of b. This is based upon the fact that BT are polynomials which, if expressed in the power basis, would have a constant term. Subtracting from this constant term the density of the isosurface to render has the effect of "shifting" the densities of the entire solid by that amount so that they line up with the zero-isosurface. This enables us to render any isosurface of a BT by simply replacing it with its "shifted" zero-isosurface pair and using the methods described in Section 3.4 to ray trace the new BT.

In order to actually find the modified weights, we have to transform the BT into a space in which one of the weights corresponds to a constant term. Inspection of equation 2 shows that Euclidean space will serve this purpose well since the value of u will be the homogeneous value in the position vector, which will always be equal

to 1 by equation 2. This will make the weight corresponding to $(i, j, k, l) = (0, 0, 0, 3)$ a constant value. By the same process as equation 3, we can write the transformed tensor by

$$\hat{\mathbf{B}}_{\beta_1,\beta_2,\beta_3} = \mathbf{T}^{-1}{}^{\alpha_1}_{\beta_1}\mathbf{T}^{-1}{}^{\alpha_2}_{\beta_2}\mathbf{T}^{-1}{}^{\alpha_3}_{\beta_3}\mathbf{B}_{\alpha_1,\alpha_2,\alpha_3} \tag{8}$$

Equation 4 shows us that the only value in the tensor which contains the weight from position $(0, 0, 0, 3)$ is $\hat{\mathbf{B}}_{3,3,3}$. Therefore, adding the density value d to this weight will result in

$$\bar{\mathbf{B}}_{\beta_1,\beta_2,\beta_3} = \begin{cases} \hat{\mathbf{B}}_{\beta_1,\beta_2,\beta_3} + d & \text{if } (\beta_1, \beta_2, \beta_3) = (3, 3, 3) \\ \hat{\mathbf{B}}_{\beta_1,\beta_2,\beta_3} & \text{otherwise} \end{cases} \tag{9}$$

Finally, transforming $\bar{\mathbf{B}}$ into screen space is given by

$$\tilde{\mathbf{B}}_{\beta_1,\beta_2,\beta_3} = \mathbf{WVP}^{-1}{}^{\alpha_1}_{\beta_1}\mathbf{WVP}^{-1}{}^{\alpha_2}_{\beta_2}\mathbf{WVP}^{-1}{}^{\alpha_3}_{\beta_3}\bar{\mathbf{B}}_{\alpha_1,\alpha_2,\alpha_3} \tag{10}$$

This process divides the transformation from equation 3 into two parts (equations 8 and 10) with the isosurface level selection (equation 9) inserted between. The result of these three equations is the screen-space, scaled tensor-form BT $\tilde{\mathbf{B}}$, which can then be substituted into equation 4 for ray tracing.

4.1.5 Rendering

In order to actually render each tetrahedral primitive, we perform a method similar to Loop and Blinn [2006]. Despite our changes to support arbitrary isosurface values, the rendering method is effectively the same. It is important to note that we only use orthogonal projection in our implementation, as it greatly simplifies a crucial portion of the rendering. Other than this one step, which will be identified as having this requirement, our system works equally well for both perspective and orthogonal projection.

We render BT Volumes in DirectX 10 using Shader Model 4.0 in order to be able to leverage geometry shaders. For this section we view a BT Volume as a tetrahedral grid since each voxel consists of some number of tetrahedra. Each point in the grid is represented with a single vertex in a vertex buffer containing its BT weights in tetrahedral form. We perform an optimization in this phase by only storing grid points which might result in an isosurface-containing tetrahedron, which we determine by looking at the BT weights. According to the BT bounding property, the minimum and maximum weights form a boundary on the density of all points inside the solid.

If both extrema are on the same side of the range considered important (which implies that this BT does not contribute to any important isosurface) the BT is thrown away. This optimization can save a significant amount of space depending on the distribution of density in the volume and demonstrates one advantage of storing our data as vertex attributes instead of textures. The actual range considered interesting is somewhat application specific. Each BT that passes this test is transformed into Euclidean space by equation 8 and into tetrahedral indexing form for compactness before being stored in its vertex. The position semantic of the vertex is set to the middle of the voxel space. The vertex buffer is interpreted as a point list during the rendering phase.

We employ a geometry buffer to expand each tetrahedron into screen-space triangles. We solve this problem efficiently by computing a triangle strip for each tetrahedron in the mapping κ which represents the screen-space triangle vertices. Because each voxel uses the same mapping κ to divide into tetrahedra, this can be done once for the entire volume and transformed into the local space of each voxel in the shader. We perform this per-frame on the CPU before any rendering is performed. This only works under orthogonal projection since perspective projection will change the screen space triangles of two different BT with the same tetrahedron number h.

Each pixel needs to know both the transformed BT tensor and the portion of the viewing ray which is inside the tetrahedron. Only intersections of the viewing ray and tensor BT which are inside the bounds of the tetrahedron count as hits, since each BT is only defined within its bounding tetrahedron. The geometry shader is responsible for calculating the extents of the tetrahedron in the z dimension at each vertex and storing these values in each vertex for interpolation. The geometry shader also unpacks the BT into tensor form, calculates the scaled tensor using equation 9, transforms the tensor into screen space using equation 10 and packs the BT back into tetrahedral form and into each output vertex for the screen-space triangles. Since the weights are equal across the entire primitive, the same values are stored at each vertex.

Our implementation also uses additional small optimizations not described here for the sake of brevity, but none of them significantly change the method or add extra limitations not described in this paper.

4.2 Reconstructing a BT Volume

Although the BT Volumes rendering algorithm is interesting as an example of geometry and pixel shaders, the fundamental question is how to generate a BT Volume which represents useful data. Section 4.2.1 describes how to approximate arbitrary continuous volumetric functions as a BT Volume using a least squares method, but a more interesting approach is to find a way of expressing the result of reconstruction filtering an arbitrary scalar volume as a BT Volume. In other words, can we find a way of representing a reconstruction filter such that discrete convolution of an arbitrary scalar-valued volume with that filter produces a BT Volume. It turns out that this is exactly what happens when the reconstruction filter itself is a BT Volume which shares a similar κ function as the resulting BT Volume. This is described in detail in Section 4.2.2.

4.2.1 Approximating an Arbitrary Volume

In order to generate a BT Volume to approximate an arbitrary scalar-valued volumetric function we first have to create a tetrahedral grid for the BT Volume which surrounds the entire function, as described in Section 4.1.2 and following. Our reconstruction method requires that these voxels overlap with the reconstructed volume voxels, so scaling is determined by the number of voxels used in the approximation. The specific mapping κ used will determine how close our approximation can be to the original function. The two main properties of κ which effect this are 1) the number of tetrahedra in the mapping and 2) the spatial distribution of tetrahedra in in the domain. Increasing the number of tetrahedra increases the representative power of the BT Volume by allowing higher frequencies to be represented. A survey of common tetrahedral grids, including many which satisfy the constraints of a κ mapping, is described by Carr et al. [2006]. Determining the optimal κ for a particular volume in terms of the balance of rendering cost and expressive power is an interesting open research problem.

Once we generate our bounding tetrahedra, we have to complete each BT in the volume by determining the optimal weights. Since BT are polynomial, a simple least-squares approach is possible. We determine a set of sample points distributed evenly in the barycentric coordinates of each bounding tetrahedron, transforming the point into Euclidean space to evaluate the target function value at each of them. We perform least-squares fitting on all BT at once in

order to ensure c^0 continuity. In addition, we artificially clamp all boundary values to zero before using a filter.

4.2.2 Using a BT Volume as a Filter

Since a volumetric reconstruction filter is simply a scalar-valued volume function, we can create a BT Volume approximation by the method from the previous section. We require that filter voxels coincide with voxels in the scalar volume. Consider a reconstruction filter, G, expressed as a BT Volume, in the formula for discrete convolution given in equation 1. Substituting $\mathbf{X} - (a,b,c)$ for $\mathbf{Q}$ in equation 7 gives the specific tetrahedron τ in the reconstruction filter as

$$\tau = BT_{\lfloor \mathbf{X} \rfloor - (a,b,c)} \left(\kappa \left(\mathbf{X} - \lfloor \mathbf{X} \rfloor \right) \right) \tag{11}$$

which can be evaluated (according to equation 2) by

$$\begin{gathered} G\left(\mathbf{X} - (a,b,c)\right) = \\ \tau(\mathbf{X} - (a,b,c)) = \\ \sum_{i+j+k+l=3} w_{ijkl} \binom{3}{ijkl} r^i s^j t^k u^l \end{gathered} \tag{12}$$

where $\mathbf{r} = (r,s,t,u) = (\mathbf{X} - (a,b,c)) \cdot (\mathbf{T})^{-1}$ and $\mathbf{T}$ is defined from the bounding tetrahedron of τ. We can substitute equation 12 for G to get

$$\begin{gathered} (A * G)(\mathbf{P}) = \\ \sum_{(a,b,c) \in C} \left(\sum_{\substack{i+j+k \\ +l=3}} w_{ijkl} \binom{3}{ijkl} r^i s^j t^k u^l \right) A\left((a,b,c)\right) \end{gathered}$$

Note that we do not have to sum over H because we defined κ not to have overlap — we only have to loop for tetrahedra with index $h = \kappa(\mathbf{X} - \lfloor \mathbf{X} \rfloor)$. Note that the tetrahedral indices i,j,k,l do not depend on the value of the outer summation. In addition, the values r,s,t,u are identical for every tetrahedron with the same h index, so they also do not depend on the outer summation. Rearranging the equation to move all terms possible out of this summation gives

$$\begin{gathered} (A * G)(\mathbf{P}) = \\ \sum_{\substack{i+j+k \\ +l=3}} \binom{3}{ijkl} r^i s^j t^k u^l \sum_{(a,b,c) \in C} \left(w_{ijkl} \cdot A\left(a,b,c\right) \right) \end{gathered} \tag{13}$$

By inspection this is the equation of a BT with the weight component equal to the entire second summation. Also note that as long as $\kappa(\mathbf{X} - (a,b,c))$ returns the same h value, which it will within a voxel, equation 13 will produce the same BT coefficients, implying that the κ function of the resulting BT Volume will be the same as the κ of the filter from equation 11.

The intuitive explanation behind this derivation is to consider the view of convolution as a sum of kernel functions centered on each sample point and weighted by the sample value. Since the volume samples are evenly spaced and κ is the same for each voxel, each tetrahedron in the result will be covered by one and only one tetrahedron from *each* piecewise BT kernel in the sum (see Figure 3). So each tetrahedron of the full volume can be expressed as a sum of BT from the kernels. BT are closed under addition, so the result of the convolution is a single BT for each tetrahedron in the full volume.

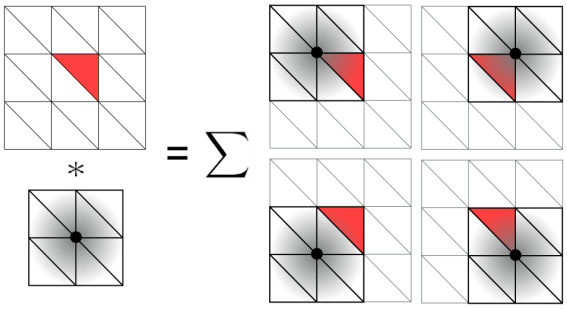

Figure 3: *We model convolution as a summation of BT solids. In order to convolve one tetrahedron of a BT Volume, pictured above in red, with a BT Volume filter with the same κ function, we need to sum together the contributions of overlapping BT in all possible translations of the filter, each scaled by the sample value at that filter position.*

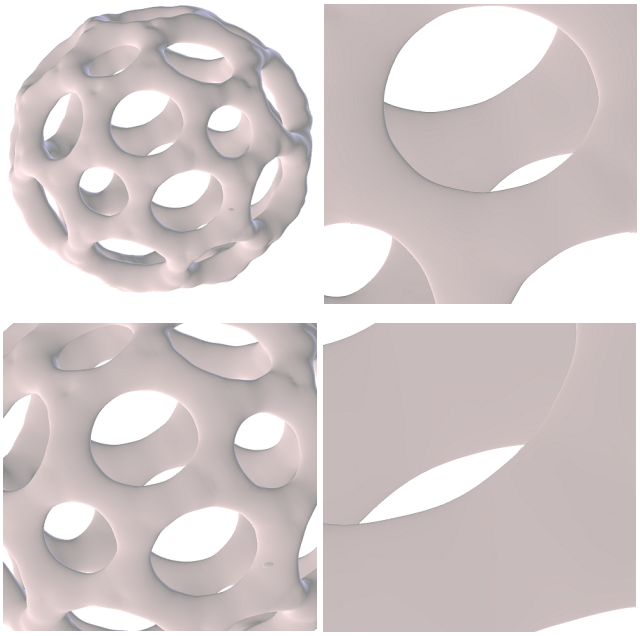

Figure 4: *Several shots in a zoom animation of a 32^3 volume reconstruction show the resolution independence of the BT Volume format.*

There is, however one important change required in order to maintain a smooth result. In equation 1 the order of voxels is reversed in the filter, but all values within a voxel are not. In order to maintain continuity between voxels, it is necessary to invert each voxel in place before evaluating equation 13. Therefore, the κ function of the resulting BT Volume will be an inverted version of the filter κ.

This is the main result from our work and allows us to represent a volume convolved with a BT filter as a BT Volume. The power of this result lies in the fact that BT volumes can be rendered in real time, thus allowing us to render high quality convolved volumes exactly, assuming we can represent the reconstruction filter as a BT Volume.

4.3 Algorithm

Our system uses all of the methods presented in this paper in a rendering algorithm as specified below.

Preprocess

- For every filter kernel, create the BT Volume approximation.
- For any filter kernel/volume pair, perform convolution to get a BT Volume.

Runtime

- For every new isosurface to render, cull out non-intersecting tetrahedra on BT Volume.
- For every frame, construct screen-space triangles in a Geometry Shader and ray trace isosurface.

5 Results

We implemented our system on top of the Direct3D 10 graphics API running on an NVIDIA 8800 GTS GPU. One feature Windows Vista provides which enhances the usability of our system is virtualization of graphics hardware memory. The space requirement of BT Volumes is very large, limiting the size volumes which can be rendered on graphics cards with less memory. Graphics memory virtualization allows us to transparently render BT Volumes which would otherwise require more space than our graphics card could support. In practice this enables rendering of large BT Volumes with a small performance hit on systems which would otherwise not have enough memory or require special paging code.

Although rendering a BT Volume is fast enough to be done in real time, computing the convolved BT Volume data is slower than we expected. Our implementation uses graphics hardware to accelerate the process, but still requires more than 30 seconds to convolve a 128^3 volume with a 6^3 filter. Note that the convolution step only has to be performed once for a given volume/filter pair, and loading a stored BT Volume into graphics memory is very fast. The remainder of this section details rendering performance and space requirements of our system.

5.1 Rendering Performance

Performance was GPU bound and the CPU under-utilized. In general, volumes up to size 64^3 run in interactive rates after optimization, while larger volumes require multiple passes to complete successfully. One optimization we performed was to cull-out all tetrahedra which did not contribute to a specific isosurface level in a geometry shader pre-pass, streaming all vertices which pass the test into a second buffer which is rendered until the isosurface changes. All images in this paper representing volumes of that size or smaller were captured from a live run of our application. The table below summarizes performance for various volumes at a screen resolution of 1600×1200 with roughly screen-filling views on both graphics cards. The two numbers given for each frames-per-second speed are the steady-state rendering speed of the slowest and fastest isosurface of a specific volume. Performance as the isosurface level changes dips slightly below these numbers because of the pre-pass which culls out all non-contributing tetrahedra.

Images of test models reconstructed and rendered using BT Volumes are shown in Figures 1,2, 4, and 5. For comparison, a Marching Cubes rendering of the 128^3 foot is shown in Figure 6.

5.2 Space Requirements

The space requirements of BT Volume data are very large. This is to be expected because of the massive amount of information being stored. BT Volumes represent a continuous function with tremendous representative capability, but this comes at a cost. Because each tetrahedron in the volume requires twenty weights to fully define a BT, and there are at least five tetrahedra in each voxel (the most compact κ consists of five tetrahedra), a naive implementation would require at least 100 times the space of a floating-point volume of the same size. For larger volumes we store 16-bit precision floats, but clearly size is still an issue. One optimization we perform is to throw away voxels which do not contain any isosurface within a range of interest. We throw away voxels which contain only isosurfaces within a small delta of zero. In practice the space saved depends completely on the volume and the size delta used. We have seen savings up to 50 percent for models with large empty spaces. The table below lists the space requirements for each of the models we rendered. Entries with a * used 16-bit precision and culled voxels contributing less than 7.8×10^{-3} to save space.

Name	Dimensions	MB	GTS-FPS
Engine*	64×128^2	127	N/A
Foot High*	128^3	187	N/A
Foot Low	64^3	135	5-8
Molecule	64^3	135	3-6
Bucky	32^3	17	10-14
Filter	6^3	.11	35-200

6 Conclusion and Future Work

In this paper we have presented a novel method for representing volumetric data in a resolution independent manner called a BT Volume. This representation is comprised of a tetrahedral grid containing Bézier Tetrahedra polynomial solids with special regularity requirements. We have demonstrated an efficient rendering algorithm for BT Volumes as well as two ways to project useful data into the BT Volume format. Rendering is achieved through the use of local ray-intersection equations on the graphics card for each tetrahedron individually. The geometry shader is used to transform each BT into screen space before rendering, as well as set the isosurface of the volume to render. The first method to create a BT Volume we presented was a simple least-squares approximation of an arbitrary volumetric function. The second method uses this least-squares technique to generate a BT Volume representation of a reconstruction filter, which we pre-convolve with an arbitrary scalar rectilinear volume. We showed how this formulation of the convolution equation results in another BT Volume which equals the reconstructed data.

Although our implementation is complete enough to act as a proof-of-concept for the rendering technique, several areas could be improved. Our least-squares approximation does not enforce C^1 continuity. In addition, because of various graphics hardware precision issues, a few pixels will incorrectly miss the isosurface being rendered. Although this artifact does appear in images in this paper, it is not as noticeable in static images as it is in motion.

In addition, we feel that there are several directions of possible future related research. So far we have only explored some interactive applications of our volume representation, but we believe that the flexibility of our system will also be useful for offline rendering. Exploring how the BT Volume representation/filtering methods presented in this paper can be enhanced further is an area which may produce useful results. Some possible directions to explore include acceleration of offline rendering for large data sets, analysis of different approximate filters, using higher order polynomial solids and/or solids of a different class (e.g., tensor product patches instead of simplex), and efficient compression/storage of BT Volume data.

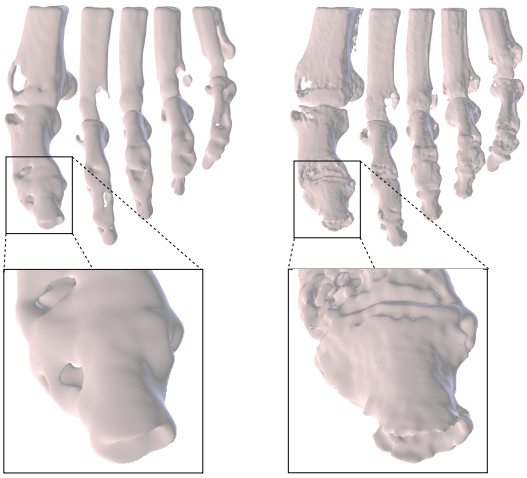

Figure 5: *Two images rendered with our system. Left:* 64^3 *foot volume isosurface corresponding to bone density. Right: The same volume at* 128^3 *resolution. Insets show how smooth the isosurface remains even at high resolution.*

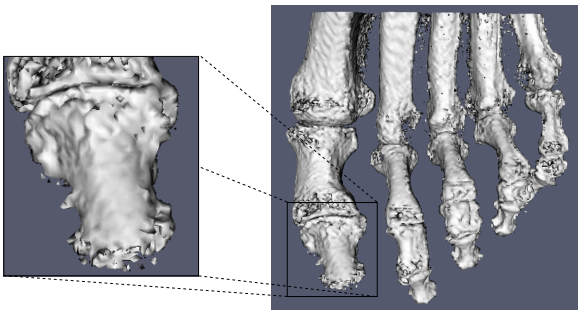

Figure 6: *Comparison image using the Marching Cubes algorithm at* 128^3 *(as implemented by ParaView), showing the rough and noisy nature of Marching Cubes renderings. Inset shows significant linear interpolation artifacts (visible as linear features in the shading and geometry).*

Acknowledgements

This work was funded in part by NSF grant 0121288. We would like to thank our anonymous reviewers for helping with the technical presentation of this paper. All volume data was taken from The Volume Library (http://www9.informatik.uni-erlangen.de/External/vollib/). We would also like to thank Dr. Alark Joshi for his support and help.

References

ANDERSON, J. C., BENNETT, J., AND JOY, K. I. 2005. Marching diamonds for unstructured meshes. In *IEEE Visualization 2005*, 423–429.

BAJAJ, C. L., CHEN, J., AND XU, G. 1995. Modeling with cubic a-patches. *ACM Trans. Graph. 14*, 2, 103–133.

CARR, H., MOLLER, T., AND SNOEYINK, J. 2006. Artifacts caused by simplicial subdivision. *IEEE Transactions on Visualization and Computer Graphics 12*, 2, 231–242.

JOHANSSON, G., AND CARR, H. 2006. Accelerating marching cubes with graphics hardware. In *CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, ACM Press, New York, NY, USA, 378.

LEVOY, M. 1990. Efficient ray tracing of volume data. *ACM Trans. Graph. 9*, 3, 245–261.

LOOP, C., AND BLINN, J. 2005. Resolution independent curve rendering using programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM Press, New York, NY, USA, 1000–1009.

LOOP, C., AND BLINN, J. 2006. Real-time gpu rendering of piecewise algebraic surfaces. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM Press, New York, NY, USA, 664–670.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 163–169.

MARSCHNER, S. R., AND LOBB, R. J. 1994. An evaluation of reconstruction filters for volume rendering. In *VIS '94: Proceedings of the conference on Visualization '94*, IEEE Computer Society Press, Los Alamitos, CA, USA, 100–107.

MITCHELL, D. P., AND NETRAVALI, A. N. 1988. Reconstruction filters in computer-graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 221–228.

PARKER, S., PARKER, M., LIVNAT, Y., SLOAN, P.-P., HANSEN, C., AND SHIRLEY, P. 1999. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (*/*), 238–250.

POLICARPO, F., OLIVEIRA, M. M., AND JO A. L. D. C. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 155–162.

RITSCHE, N. 2006. Real-time shell space rendering of volumetric geometry. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, ACM Press, New York, NY, USA, 265–274.

ROSSL, C., ZEILFELDER, F., NURNBERGER, G., AND SEIDEL, H.-P. 2003. Visualization of volume data with quadratic super splines. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society, Washington, DC, USA, 52–60.

TATARCHUK, N. 2006. Dynamic parallax occlusion mapping with approximate soft shadows. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 63–69.

THEISEL, H. 2002. Exact isosurfaces for marching cubes. In *Computer Graphics Forum*, Blackwell Publishers for Eurographics Association, 19–31.

TREECE, G. M., PRAGER, R. W., AND GEE, A. H. 1999. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics 23*, 4, 583–598.