Enhancing the Interactive Visualization of Procedurally Encoded Multifield Data with Ellipsoidal Basis Functions

Yun Jang†, Ralf P. Botchen‡, Andreas Lauser‡, David S. Ebert†, Kelly P. Gaither§, Thomas Ertl‡

Purdue University Rendering and Perceptualization Lab, Purdue University, USA †
Visualization and Interactive Systems, University of Stuttgart, Germany ‡
Texas Advanced Computing Center, University of Texas at Austin, TX, USA §

Abstract
Functional approximation of scattered data is a popular technique for compactly representing various types of datasets in computer graphics, including surface, volume, and vector datasets. Typically, sums of Gaussians or similar radial basis functions are used in the functional approximation and PC graphics hardware is used to quickly evaluate and render these datasets. Previously, researchers presented techniques for spatially-limited spherical Gaussian radial basis function encoding and visualization of volumetric scalar, vector, and multifield datasets. While truncated radially symmetric basis functions are quick to evaluate and simple for encoding optimization, they are not the most appropriate choice for data that is not radially symmetric and are especially problematic for representing linear, planar, and many non-spherical structures. Therefore, we have developed a volumetric approximation and visualization system using ellipsoidal Gaussian functions which provides greater compression, and visually more accurate encodings of volumetric scattered datasets. In this paper, we extend previous work to use ellipsoidal Gaussians as basis functions, create a rendering system to adapt these basis functions to graphics hardware rendering, and evaluate the encoding effectiveness and performance for both spherical Gaussians and ellipsoidal Gaussians.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Scientific Visualization, Ellipsoidal Basis Functions, Functional Approximation, Texture Advection

1. Introduction
Most current visualization techniques are datagrid specific and do not allow scientists and researchers to interactively visualize various unstructured and scattered datasets in a single system on their desktop computers. As one solution, Jang et al. [JWH’04] and Weiler et al. [WBS’05] developed an interactive visualization and feature detection system for scalar, vector and multifield data using radial basis functions (RBFs). RBFs have been widely used as basis functions to approximate datasets, and the function is constructed as sums of RBFs. Mathematically, for given data samples \( P_j = (\bar{x}_j, Y_j) \), \( j = 1, ..., N \), where the values \( \bar{x}_j \) are the spatial locations and the values \( Y_j \) are the data values that exist at the corresponding spatial locations, the data values can be approximated with a function \( f(\bar{x}) \) defined as

\[
f(\bar{x}) = \sum_{i=1}^{L} \lambda_i \phi \left( \bar{x}, \bar{\mu}_i, \sigma_i^2 \right),
\]

where \( L \) is the number of basis functions, \( \lambda_i \) the weight, \( \mu_i \) the center, and \( \sigma_i^2 \) the variance of a single basis function. With this mathematical formula, \( \lambda, \mu \) and \( \sigma^2 \) are optimized to find the best approximation of the original data values.

Jang et al. [JWH’04] and Weiler et al. [WBS’05] used spatially-limited spherical Gaussian RBFs, since the truncated radially symmetric basis functions are quick to evaluate and simple for encoding optimization. However, spherical Gaussians are not the most appropriate choice for volumetric data that is not radially symmetric and they are espe-
2. Related Work

Radial basis functions (RBFs) are radially symmetric functions and have been widely used for surface and functional approximations due to the symmetric property in broad research areas, including surface modeling, geometric modeling, spatial interpolation in GIS applications, and visualization of 3D scattered data [Har90, SPOK95, MM99, Gos00, CBC*01, TO02, ZRP02].

Researchers have also proposed several different polynomial functions to approximate the construction of soft objects [WMW86], or solid geometry applied to implicit surfaces [KS97]. Recently, Chen [Che05] compared several different radial basis functions that are suited to combine point clouds and conventional volume objects. Jang et al. [JWH*04] and Weiler et al. [WBS*05] presented techniques for spatially-limited spherical Gaussian radial basis function encoding and visualization of volumetric scalar, vector, and multifield datasets.

To provide a greater variety of functional approximation and modeling options, researchers proposed to use ellipsoids instead of RBFs. Ellipsoids have been used in many research areas, such as segmentation [OCK05], object detection [GS00], and filtering of noisy data [SBS05]. In the area of data fitting, Calafiore [Cal02] proposes an approximation of n-dimensional data using ellipsoidal primitives and showed that a "distance-of-squares" geometric error criterion gives stability for Gaussian noise. This approach, however, is limited to several hundreds of points because of numerical problems. Li et al. [LWP*04] proposed a fitting method using ellipsoids for implicit surfaces with a limited number of data points. Recently Huang et al. [HFBC06] showed a shape-based approach for thin structure segmentation and visualization in biomedical images using an ellipsoidal Gaussian model.

A common approach to visualize both structured and scattered flow data is texture advection. The basic idea is to represent a dense aggregation of particles, or a sparse, but larger dye pattern by a texture and to transport them along the underlying steady flow, as proposed in [CL93, MB95]. More recently published work extended this concept to unsteady 2D flows [JEH00, vW02]. A different related technique has been introduced that addresses the problem of visualizing uncertainty in flow [BWE05]. A hybrid technique was also proposed by Weiskopf [WE04] that can be applied to curved surfaces, and evaluates 3D vector fields [LBS03]. An overview of the state-of-the-art in texture based flow visualization techniques is given in Laramee et al. [LHD*04].

3. Spherical and Ellipsoidal Gaussians

As previously mentioned, the radial basis function (RBF) is a radially symmetric function where the RBF volume has a spherical shape. For volume visualization, several researchers [JWH*04, WBS*05] have used spherical Gauss-
sions as basis functions. The spherical shaped basis function, however, has a limitation in fitting long, high-gradient shapes, for example cylindrical shapes. The radius might reach the shortest boundary of the area and require many small RBFs to fit one long shape. The left image in Figure 1 visually displays this artifact for spherical RBFs.

In order to reduce these artifacts, we change the spherical Gaussians to ellipsoidal Gaussians, which are ellipsoidal basis functions (EBF). There are two kinds of ellipsoidal Gaussians: axis-aligned and arbitrary directional EBFs. The middle image in Figure 1 is the result of using the axis-aligned ellipsoidal Gaussian. As seen in Figure 1, the narrow shape is well represented by the ellipsoidal Gaussian. Additionally, only a small number of cells is needed for rendering when using the octree data structure for spatial subdivision. Cuboids are used to generate the spatial cell distribution rather than cubes and they are aligned according to the influences along the axes. A comparison of these three basis functions is shown in Figure 2. This Figure shows a long diagonal data distribution and the influences of the three basis functions are drawn overlaid on the data.

Using the Mahalanobis distance, the ellipsoidal Gaussian basis function in 3D space can be represented in a matrix form as follows:

\[
\phi(\vec{x}) = \exp \left( -\frac{1}{2} (\vec{x} - \vec{\mu})^T V^{-1} (\vec{x} - \vec{\mu}) \right)
\]  

(2)

\(V^{-1}\) is positive definite and is defined by a rotation matrix, \(R\), and a scalar matrix, \(S\), as following:

\[
V^{-1} = \begin{bmatrix}
  a_1 & a_4 & a_6 \\
  a_4 & a_2 & a_5 \\
  a_6 & a_5 & a_3
\end{bmatrix} = R \cdot S^{-1} \cdot S^{-1} \cdot R^T
\]

\(\vec{x}\) is a coordinate vector, \([x \ y \ z]^T\) and \(\vec{\mu}\) is the center vector of an ellipsoidal Gaussian, \([\mu_x \ \mu_y \ \mu_z]^T\). Moreover, for a rotation matrix, \(R^{-1} = R^T\) and for a scaling matrix \(S^T = S\).

Since all rotation angles are set to zero for the axis aligned ellipsoidal Gaussians, off-diagonal components in \(V^{-1}\) are zero and diagonal components are positive, \(a_1 = \frac{1}{\sigma_x^2}, a_2 = \frac{1}{\sigma_y^2}, a_3 = \frac{1}{\sigma_z^2}\), and \(a_4 = a_5 = a_6 = 0\). Therefore, the axis aligned ellipsoidal Gaussian can be represented as

\[
\phi(x, y, z) = \exp \left\{ -\frac{(x - \mu_x)^2}{2\sigma_x^2} - \frac{(y - \mu_y)^2}{2\sigma_y^2} - \frac{(z - \mu_z)^2}{2\sigma_z^2} \right\}
\]  

(3)

On the other hand, in the arbitrary directional ellipsoidal Gaussian, off-diagonal components may not be zero. Therefore, we consider components of \(S\) and \(R\) instead of the direct form, Equation 3, to make sure that \(V^{-1}\) is positive definite.

4. Functional Approximation using Ellipsoidal Gaussians

Functional approximation is used in this paper to find the best parameters of the basis functions to fit the data values as closely as possible. In Section 1, a functional approximation using RBFs is presented. For the extension of RBFs to EBFs, we use the covariance \(V\) rather than the single variance in the functional approximation. The function using \(V\) is redefined as

\[
f(\vec{x}) = \sum_{i=1}^{L} \lambda_i \phi(\vec{x}, \vec{\mu}_i, V_i)
\]  

(4)

The parameters include centers (\(\mu\)), covariances (\(V\)), and weights (\(\lambda\)). Our approach to the functional approximation is based on previous RBF work [JWH’04, WBS’05]. However, EBF parameters are optimized in this work and the number of parameters is larger than the number of RBF parameters. In the 3D scalar approximation, we add 2 more parameters for the axis aligned ellipsoidal Gaussians and 5 more parameters for the arbitrary directional ellipsoidal Gaussians, compared to spherical Gaussians. For 3D vector approximation, 6 more parameters are needed for the axis aligned ellipsoidal Gaussians and 15 more parameters for the arbitrary directional ellipsoidal Gaussians.

For an algorithm to find the best parameters for the approximation, we use nonlinear optimization. In the algorithm, we quickly compute reasonable initial starting parameters using the given data (Section 4.1). This provides better results in the nonlinear optimization and we insert these initial parameter values into the main optimization algorithm (Section 4.2). In the algorithm, we find the parameters for only a single basis function at each iteration. Once the parameters that reduce the approximation error the most are found, we remove the influence of the basis function from the data values and compute the residual functional values (errors) using either the L2-norm or the H1-norm (Section 4.3). These residuals are then used in the next iteration. We iterate the above approximation system until our error tolerance is satisfied. We use an error of 5% of the maximum data value as our error tolerance. However, RMS errors are typically less than 1-3% in the resulting datasets. Once the best functional approximation is found, we divide the volume into several small subvolumes using an adaptive oc-
tree structure based on the basis function distribution (Section 4.4). This allows us to improve rendering speed. Moreover, this algorithm can be applied to both scalar and vector datasets (Section 4.5).

4.1. Initial Parameter Approximation

The initial starting parameter values for nonlinear optimization are important for the optimization convergence. In this section, we describe how we compute the initial starting EBF parameter values.

The starting center value is set to the maximum data value point and the weight is set to the data value at this center. Since we compute residuals after every iteration, the residuals become the new data values in the next iteration. Therefore, the maximum error point is the maximum value data point. The position of this data point is set to the center of the basis function and the maximum value at the point is set to the weight of the basis function for both RBF and EBF.

For RBF variances, the data values are used and the variance is written as

\[ \sigma_{ij}^2 = \frac{r^2}{2 \ln \left( \frac{|Y_i|}{Y_j} \right)} , \quad j = 1, \ldots, N, \]  

where \( Y_i \) is the data value at the center and \( Y_j \) is the value at the \( j \)th data point assuming that \( \text{sgn}(Y_i) = \text{sgn}(Y_j) \) and \( |Y_i| > |Y_j| \). \( r \) is defined as \( |x_j - \mu_i| \) and \( N \) is the number of points. We then average \( \sigma_{ij}^2 \) for \( i = 1, \ldots, N \) to compute the approximate variance.

However, the variances of axis aligned EBFs are computed using both data values and gradients since there is more than one unknown variance in the axis aligned EBF. By taking derivatives of Equation 3, we approximate the direction variance as follows:

\[ \sigma_{xj}^2 = -(x_j - \mu_{xj}) \cdot \frac{Y_j}{dy_j/dx_j} , \quad j = 1, \ldots, N \]  

The y and z direction variances can be computed similarly. We then average these variances for the approximate variances.

For the arbitrary directional EBFs, we use the above approximated variances and set the off-diagonal components to zero, which means that all rotation angles are zero, for the initial parameter values for the nonlinear optimization.

4.2. Nonlinear Optimization

Once all approximate parameters are computed, we insert the parameters into the nonlinear optimization algorithm. In this work, we use the Levenberg Marquardt [MNT99] approach to minimize sum squared error. In this approximation algorithm, we optimize all parameters at the same time. We optimize the parameters of the axis aligned EBFs in the same way as we optimize RBFs, since off-diagonal components in the covariance matrix are all zero. However, for arbitrary directional EBFs, we optimize the scaling components in the scaling matrix and the angles in the rotation matrix instead of optimizing \( a_1, \ldots, a_6 \) for 3D datasets since the result from optimization of \( a_1, \ldots, a_6 \) might not construct a real covariance matrix.

4.3. Error Measurement

Even though EBFs are good basis functions for any structural dataset after optimization, it can still show visible artifacts. Therefore, we use two different cost functions in the optimization and compare these two cost functions. We normally use the L2-norm based cost function (Equation 7) which only uses the data values. For increased visual accuracy, we pre-approximate gradients at each data point and add them to the cost function, using the H1-norm based cost function (Equation 8) [GE98] in order to reduce the visual artifacts. Comparison results between these two cost functions are shown in Figure 8 and 9.

\[ \psi = \frac{1}{2} \sum_{j=1}^{N} (f(x_j) - Y_j)^2 \]  

\[ \psi = \frac{1}{2} \sum_{j=1}^{N} \left\{ (f(x_j) - Y_j)^2 + \frac{\partial f(x_j)}{\partial x} - \frac{\partial f(x_j)}{\partial x} \right\} \]  

4.4. Volume Subdivision for Rendering

In order to render RBFs and EBFs faster, we utilize an adaptive octree volume subdivision based on the influence of RBFs and EBFs. For RBFs, influence ranges are the same in all directions, but EBFs have different influence ranges in different directions. Figure 2 shows the influence ranges of RBFs and EBFs. The influence radii of RBFs are computed as

\[ r_i = \sigma_i \cdot \sqrt{2 \cdot \ln \left( \frac{|\lambda_i|}{\varepsilon} \right)} \]  

where \( \varepsilon \) is a user defined error tolerance. Therefore, subdivided volumes are cubes. For the axis aligned EBFs, the influence range in x direction can be computed similarly as

\[ r_x = \sigma_x \cdot \sqrt{2 \cdot \ln \left( \frac{|\lambda_x|}{\varepsilon} \right)} \]  

The influence ranges in y and z direction can be computed similarly to the above. Note, that the influence range is a cuboid, but not just a cube. The influence range of the arbitrary directional EBF in 3D can be computed in each direction as follows:

\[ r_{x_i} = \frac{a_2 a_3 - a_2^2}{|V - 1|} \cdot 2 \cdot \ln \left( \frac{|\lambda_{x_i}|}{\varepsilon} \right) \]  

\[ r_{y_i} = \frac{a_2 a_4 - a_2^2}{|V - 1|} \cdot 2 \cdot \ln \left( \frac{|\lambda_{y_i}|}{\varepsilon} \right) \]  

\[ r_{z_i} = \frac{a_2 a_5 - a_2^2}{|V - 1|} \cdot 2 \cdot \ln \left( \frac{|\lambda_{z_i}|}{\varepsilon} \right) \]
\[ r^2 = \frac{a_1 a_2 - a_3^2}{|V^{-1}|} \cdot 2 \cdot \ln \left( \frac{\lambda_i}{\varepsilon} \right) \]

Note, this influence range is also a cuboid, but not just a parallelepiped.

To speed rendering, we use the volume subdivision algorithm proposed by Jang et al. \[ \text{JWH'04}\]. We can obtain the distribution of basis functions using the influence ranges and can divide a volume into smaller subvolumes by setting a maximum number of basis functions rendered in a subvolume using an adaptive octree. Therefore, each subvolume contains less than the maximum number of basis functions, increasing rendering performance.

### 4.5. Scalar vs. Vector Encoding

Scalar data approximation is performed using the above approach since there is only one value at each data point. Therefore, the maximum data point is set to be a center of an ellipsoidal Gaussian and its value at the point is set to be a weight of the ellipsoidal Gaussian. Variance terms are approximated using either Equation 5 or 6 according to the basis functions. We then optimize all these parameters using Levenberg Marquardt optimization. For the 3D spherical Gaussian, 5 parameters (center, weight, and variance) are optimized at the same time, 7 parameters for the axis aligned ellipsoidal Gaussians, and 10 parameters for the arbitrary directional ellipsoidal Gaussians.

While scalar data can be approximated simply, vector data needs further treatment in order to optimize all vector components at the same time. Since we choose one center with different weights and different variances between vector components, we set the center at the point that has the maximum L2 norm of the vector and each weight of each vector component to be its own vector component value. We then compute approximate variances for each vector component similar to the above algorithm. Afterwards, we optimize all approximated parameters at the same time. In 3D, there are 9 parameters for the spherical Gaussians, 15 parameters for the axis aligned ellipsoidal Gaussians, and 24 parameters for the arbitrary directional ellipsoidal Gaussians.

Once vector data encoding is complete, we subdivide the volume according to the maximum influence vector component; otherwise, we might lose some of the vector component.

### Table 1: Statistical functional approximation results. Each number indicates the number of basis functions for RBFs (I), axis aligned EBFs (II) and arbitrary directional EBFs (III).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>I</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marschner Lobb</td>
<td>2,092</td>
<td>208</td>
<td>112</td>
</tr>
<tr>
<td>Convection(70th)</td>
<td>237</td>
<td>101</td>
<td>90</td>
</tr>
<tr>
<td>Bluntfin</td>
<td>891</td>
<td>264</td>
<td>282</td>
</tr>
<tr>
<td>Oil Reservoir</td>
<td>59</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Water Channel (4th)</td>
<td>895</td>
<td>293</td>
<td>299</td>
</tr>
</tbody>
</table>

For EBF-based volume visualization, we use a viewpoint slicing approach to reconstruct the volume data for each fragment. The major difference to standard slice based techniques is due to our cell based data structure, which is explained in section 5.1. The basic principle of our rendering system is shown in Figure 3. All necessary information, like the coefficients of the basis functions, are stored in 2D texture maps and get uploaded to the texture memory of the graphics card. For each pre-computed slice polygon, the volume values are also reconstructed per rasterized fragment, depending on the chosen algorithm, and blended into the frame buffer with back-to-front blending.

### 5. GPU-based Rendering

With the continuing evolution and improvement in performance of graphics hardware, slice based volumetric rendering techniques that use GPU-support, have positioned themselves as an efficient tool for imaging and the visual analysis of scalar and multivariate fields.

For EBF-based volume visualization, we use a viewpoint slicing approach to reconstruct the volume data for each fragment. The major difference to standard slice based techniques is due to our cell based data structure, which is explained in section 5.1. The basic principle of our rendering system is shown in Figure 3. All necessary information, like the coefficients of the basis functions, are stored in 2D texture maps and get uploaded to the texture memory of the graphics card. For each pre-computed slice polygon, the volume values are also reconstructed per rasterized fragment, depending on the chosen algorithm, and blended into the frame buffer with back-to-front blending.

Several different rendering techniques that use GPU-based reconstruction of encoded multfield volumetric data, like volume-rendered isosurfaces, shock detection, vortex detection and helicity computation have been introduced in \[ \text{WBS'05}\]. In this work, we present our new EBF-based visualization techniques for those feature extraction methods with a focus on texture based flow visualization. Further, we show their benefits in accuracy of reconstruction and performance in rendering compared to radial basis functions.

### 5.1. Data Structure and Texture Layout

The rendering speed of the system highly depends on the number of basis functions that have to be decoded per frag-
ment. To expedite this process, we exploit the benefits of the following two facts. First, reducing number of basis functions, without losing quality, has a positive effect on the rendering time. We achieve this reduction of basis functions by our newly proposed encoding, as described in Section 3. Second, the volume expansion of most basis functions does not cover the whole volume and, therefore, only influences certain fragments. We use a hierarchical octree-like space partitioning. For each octree-cell, a subset of basis functions is stored, namely, only those which affect the volume enclosed by the cells boundary. Intersecting each slice polygon with all cells, and then rasterizing, in the fragment processing stage, for each fragment the appropriate cell is known and can be directly accessed in the texture. This highly reduces the number of EBFs that have to be evaluated per fragment.

An elaborate texture layout was designed to store the coefficients of all basis functions successively per cell, so that it can be easily accessed during the fragment processing stage. For every cell, an index points to the location of the first basis function in the texture. A second index stores the number of basis functions contained in the cell, and thus triggers the number of dynamic loops inside the shader. The substantial layout as described in [JWH*04] was maintained and extended to fit the need of storing more parameters for the arbitrary directional EBFs. With vector data that has been encoded with arbitrary directional EBFs, a set of eight textures is needed to store all parameters, as illustrated in Figure 3. In particular, six RGB textures are needed for all covariance matrix elements $a_{i,j}$, One RGB texture is used to store all centers $\mu_i$ and one RGB texture is used for the weights $\lambda_i$.

In the rendering stage, back-to-front blending is applied. Due to that we can not render all cells consecutively, since this would lead to blending artifacts, because the cells are not stored as a depth sorted list. Even if they were, it would still lead to artifacts on cell boundaries. A better way is to render slice by slice, since the exact depth and order of all slices is known, so we only need to compute the coordinates of the intersection polygon for each slice. For each intersection polygon, the fragment shader is called with a pointer to the first basis function in the texture and the number of functions to evaluate. Although the hierarchical structure reduces the number of evaluated functions, and thus improves rendering speed, it can also slow down the rendering if the hierarchical subdivision level is chosen too high. This effect is a result of the small call overhead that is needed for the render call of every intersection polygon. For too many intersection polygons, this exceeds the gain of reducing the amount of evaluated basis functions.

5.2. Texture-based Flow Visualization

For a global visualization of flow fields, texture advection is a well suited and common technique, that is capable of representing the underlying flow with a low density up to a high density of particles. The transport is based on a semi-Lagrangian scheme, which is briefly described below.

The particles that are injected into the flow are represented on a regular grid, namely a texture, which is denoted as the property field $\rho(\vec{x})$. From the Eulerian point of view, the position of a particle is implicitly given by the location of the corresponding texel in the property field. Particles are transported along streamlines for steady, or along pathlines for unsteady vector fields. Therefore, the Lagrangian formulation of the underlying equation of motion,

$$\frac{d\vec{x}(t)}{dr} = \vec{v}(\vec{x}(t), t),$$

can be integrated to compute the pathline of an advected massless particle,

$$\vec{x}(t_1) = \vec{x}(t_0) + \int_{t_0}^{t_1} \vec{v}(\vec{x}(t), t) \, dr,$$

where $\vec{v}(\vec{x}(t), t)$ is the vector field and $t$ denotes time. This equation can be used to compute the semi-Lagrangian transport as in [JEH01, Sta99]. Starting from the current time step $t_0$, an integration backwards in time according to Eq. (9) provides the position at the previous time step $\vec{x}(t_0 - \Delta t)$. Therefore, first-order Euler integration typically provides sufficient accuracy:

$$\vec{x}(t_0 - \Delta t) = \vec{x}(t_0) - \Delta t \vec{v}(\vec{x}(t_0), t_0).$$

If higher accuracy is needed, more sophisticated, higher-order integration schemes, e.g. Runge-Kutta can be similarly utilized. The property field $\rho(\vec{x})$ is evaluated at this previous location to access the particle that is transported to the current position. It can be implemented as a backward texture lookup to compute particle positions at a previous time step $x(t - \Delta t)$. First order Euler integration leads to

$$\rho(\vec{x}(t_0), t_0) = \rho(\vec{x}(t_0 - \Delta t), t_0 - \Delta t).$$

Bilinear interpolation is applied to reconstruct the property.
The implementation of the rendering framework is based on C++ and OpenGL, and runs on a standard desktop PC with graphics hardware that supports dynamic loops. All timings for the datasets presented in Table 2 have been obtained on a Pentium 4 3400MHz processor with an NVIDIA GeForce 7800 GTX graphics board. We have tested our approach on a variety of datasets. The oil reservoir data was computed by the Center for Subsurface Modeling at The University of Texas at Austin. This 156,642 tetrahedra dataset is a simulation of a black-oil reservoir model used to predict placement of water injection wells to maximize oil from production wells. The Marschner-Lobb data was obtained using an equation developed by Marschner and Lobb [ML94], and we used the same parameters that they used except for spatial range. In this work, we sampled 50,000 points randomly in $-0.5 < x, y, z < 0.5$. The natural convection dataset simulates a non-Newtonian fluid in a box, heated from below, cooled from above, with a fixed linear temperature profile imposed on the side walls. The simulation was developed by the Computational Fluid Dynamics Laboratory at The University of Texas at Austin and was run for 6000 time steps on a mesh consisting of 48000 tetrahedral elements. The Bluntfin dataset was developed by C.M. Hung and P.G. Bunin and it is a 40x32x32 single-zone, curvilinear, structured block dataset in plot3D format. The water channel dataset is

### Table 2: Performance measurements for different scenarios in frames per second (fps). All volumes have been rendered with 128 slices and a viewport of 512².

<table>
<thead>
<tr>
<th>Dataset</th>
<th>I</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marschner-Lobb</td>
<td>0.1</td>
<td>0.7</td>
<td>0.9</td>
</tr>
<tr>
<td>Convection</td>
<td>0.7</td>
<td>1.3</td>
<td>1.1</td>
</tr>
<tr>
<td>Bluntfin</td>
<td>1.3</td>
<td>1.8</td>
<td>1.6</td>
</tr>
<tr>
<td>Oil Reservoir</td>
<td>2.7</td>
<td>10.6</td>
<td>8.0</td>
</tr>
<tr>
<td>Water Channel (4th)</td>
<td>44.8</td>
<td>128.4</td>
<td>87.2</td>
</tr>
</tbody>
</table>
Figure 7: Comparison of two time steps (70th : a,c,e, 150th : b,d,f) of the natural convection datasets encoded with RBFs (a) and (b), axis aligned EBFs (c) and (d), and arbitrary directional EBFs (e) and (f). Important areas are highlighted with white boxes for a better comparison.

Figure 8: Comparison of the Marschner-Lobb dataset encoded with RBFs (a), RBFs with gradient (b), axis aligned EBFs (c), axis aligned EBFs with gradient (d), arbitrary directional EBFs (e) and arbitrary directional EBFs with gradient (f).

We compare our encoding results according to basis functions in Table 1. As shown in the table, EBFs generate better statistical results than RBFs. Since all datasets have non-spherically shaped volumes, EBFs are more flexible and appropriate to approximate the given volumes. For the comparison between axis aligned EBFs and arbitrary directional EBFs, the results depend on the datasets. If one dataset has more diagonal shapes than another, the arbitrary directional EBF is a more appropriate basis function. Otherwise, the axis-aligned EBF is more appropriate as approximation using arbitrary directional EBFs requires more computation than approximation using axis-aligned EBFs and there are more parameters required for rendering system. Moreover, in hierarchical volume subdivision, the influence range of arbitrary directional EBFs may be wider than the influence range of axis aligned EBFs since an adaptive octree is used for the volume subdivision and the octree is based on a cuboid, not a parallelepiped.

To give a fair visual comparison of the encoding techniques, the data has been encoded by all methods, with approximately the same cost, according to the error criteria. Therefore, we get an impression of how the number of basis functions needed by each technique impacts the quality of the reconstructed structures. Figure 7 shows a comparison using two time steps of the natural convection data. Since it has mostly axis aligned symmetric shapes, axis aligned EBFs and arbitrary directional EBFs show similar statistical and visual results. However, the result using RBFs shows ar-
Figure 9: Comparison of the Bluntfin dataset encoded with RBFs (a), RBFs with gradient (b), axis aligned EBFs (c), axis aligned EBFs with gradient (d), arbitrary directional EBFs (e) and arbitrary directional EBFs with gradient (f).

Figure 8 shows the reconstructions of the Marschner Lobb dataset from six diverse encodings. The left column shows three rendering results using the L2-norm based cost function. Since the Marschner Lobb data has very high frequency data values, the RBF result shows the worst approximation and EBFs show better approximation. The right column shows the rendering results using the H1-norm based cost function. The H1-norm based cost function gives more accurate results compared to the L2-norm based cost function.

In Figure 9, we compare three rendering results for the Bluntfin dataset generated with H1-norm based cost function. Similar to the Marschner Lobb result, EBFs show better results than RBFs. As shown in Figure 7 and 9, the arbitrary directional EBF encoding provides the best reconstruction with fewest basis functions. Even though this fragment shader needs the most instructions to evaluate Eq. 2, the minimum number of basis functions leads to the best performance.

Texture based flow visualization was applied to one slice of the water channel dataset in Figure 4. Although all three different techniques show very similar results in this case, the axis aligned encoding needs fewer basis functions and, thus, gives superior real-time rendering results. Further, the advection technique in combination with the arbitrary slice plane enables the user to interactively explore flow features as shown for the tornado dataset in Figure 6. Performance of our rendering system is presented in Table 2. As shown in the table, generally EBFs give better performance results in all datasets.

7. Conclusion
We have presented a functional approximation to scattered volumetric data using RBFs and EBFs. By extension to EBFs, we can overcome the approximation and visualization problems common for non-spherical structures. In the paper, we have shown both statistical and visual results for comparison. As shown previously, EBFs are more flexible and appropriate for any data distribution, and EBFs give greater compression and more accurate visual representation of datasets. Although the extension to EBFs gives us better approximation and visual accuracy, visual artifact can still be visible for some datasets. In order to reduce these artifacts, we have used both the L2-norm based and the H1-norm based cost function to examine their influence to optimization and visualization. We have presented a comparison of cost functions and shown that the H1-norm based cost function provides better visual representations than the L2-norm based cost function. Finally, we have presented texture advection techniques using the encoded data and we have shown improved vector field visualization results using EBF encoding.

In future work, we will investigate various cost functions for the encoding and examine the visual and statistical accuracy of these metrics. In addition, since the H1-norm based cost function requires significant computation, we will optimize our algorithm to reduce the amount of computation. We will also examine the scattered data distribution to select the most appropriate data-specific basis function for better approximation and more accurate visualization.

Acknowledgements
We wish to thank the reviewers for their suggestions and helpful comments. We also thank NVIDIA for donating graphics hardware. This work has been supported by the US National Science Foundation under grants NSF ACI-0081581 and NSF ACI-0121288.

References
Y. Jang, R. P. Botchen, A. Lauser, D. S. Ebert, K. Gaither, T. Ertl / EBF


© The Eurographics Association and Blackwell Publishing 2006.