Towards a High-quality Visualization of Higher-order Reynold's Glyphs for Diffusion Tensor Imaging

Mario Hlawitschka, Younis Hijazi, Aaron Knoll, and Bernd Hamann

Abstract Recent developments in magnetic resonance imaging (MRI) have shown that displaying second-order tensor information reconstructed from diffusion-weighted MRI does not display the full structure information acquired by the scanner. Therefore, higher-order methods have been developed. Besides the visualization of derived structures such as fiber tracts or tractography (directly related to stream lines in fluid flow data sets), an extension of Reynold's glyph for second-order tensor fields is widely used to display local information. At the same time, fourth-order data becomes increasingly important in engineering as novel models focus on the change in materials under repeated application of stresses. Due to the complex structure of the glyph, a proper discrete geometrical approximation, e.g., a tessellation using triangles or quadrilaterals, requires the generation of many such primitives and, therefore, is not suitable for interactive exploration. It has previously been shown that those glyphs defined in spherical harmonic coordinates can be rendered using hardware acceleration. We show how tensor data can be rendered efficiently using a similar algorithm and demonstrate and discuss the use of alternative high-accuracy rendering algorithms.

Mario Hlawitschka and Bernd Hamann
Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, University of California, Davis, CA, e-mail: {hlawitschka | bhamann}@ucdavis.edu

Younis Hijazi
Frauenhofer ITWM and University of Kaiserslautern, Germany e-mail: younis.hijazi@itwm.frauenhofer.de

Aaron Knoll
University of Kaiserslautern, Germany e-mail: aknoll@uni-kl.de

1 Introduction

When implementing and testing novel visualization techniques, basic methods for displaying data are important to verify their correctness. One of the best-known techniques is the visualization of glyphs, i.e., small icons representing the local data values. Whereas in vector visualization a single direction indicated by an arrow can be used to display the local information, second-order information can be represented by displaying scaled eigenvectors derived from the tensor's matrix representation. Even though this representation displays all information, surface glyphs are often preferred as they show the continuous behavior and, in general, reduce visual clutter: Spheres spanned by the scaled eigenvectors are the most general representation for positive-definite symmetric second-order tensor fields, but a generalization to higher-order tensors is hard to derive. Therefore, the Reynold's glyph for second-order tensor fields has been extended to higher-order data.

For second-order tensors, the representation of the glyph is straightforward and can be implemented by sampling a sphere and scaling the radius according to the function

$$f(\mathbf{x}) = \mathbf{x}^T D \mathbf{x}, \tag{1}$$

where $\mathbf{x}$ is a unit vector and D the tensor's matrix representation. Rewriting the equation using Einstein's sum convention, the extension to higher-order tensors becomes obvious [6] and is given by

$$f(\mathbf{x}) = T_{i_1 i_2 i_3 \ldots i_n} \mathbf{x}_{i_1} \mathbf{x}_{i_3} \mathbf{x}_{i_3} \cdots \mathbf{x}_{i_n}. \tag{2}$$

where the sum is implicitly given over same indices. The standard way of rendering those glyphs is by sampling a tessellation of a sphere. The two most common methods used to display higher-order glyphs are sampling the glyph along the azimuthal and longitudinal coordinates of a sphere, which leads to an unbalanced distribution of sampling points close to the poles, and sampling the glyph using a subdivision of basic shapes, usually triangulated platonic solids (tetrahedra, octahedra, and icosahedra). Applying those subdivision schemes produces several hundred triangles per glyph and, when displaying slices of the data set with several hundreds of glyphs, the increasing memory consumption negatively influences the performance of the whole visualization system. Even though the described method introduces an almost uniform sampling on the sphere, it does not provide a uniform sampling on the surface, which should be sampled depending on the curvature of the glyph, i.e., a refined sampling where large curvatures occur and a coarse sampling in flat areas. While increasing the smoothness of the glyph's representation, this method leads to an increased computational complexity. Given the fact that the function f relates to the spherical harmonic representation, which can be seen as a Fourier transform on the sphere, it can be shown that higher-order tensors introduce more high-frequency components on the surface that require finer tessellation. The increasing angular resolution of diffusion-weighted magnetic resonance scans and the increasing angular precision provided by post-processing tools require the data to be represented

at order eight to twelve (cf. Tournier et al. [20]) which exceeds the point where the generation of geometry is no longer reasonable.

We review recent work on higher-order glyph visualization and show how to apply these techniques efficiently to tensor data. Several issues arise with this technique when moving to higher-order representations that we resolve. We propose alternative rendering schemes for high-quality rendering.

2 Related Work

Glyph rendering has a long history in visualization. With the raise of modern graphics boards, hardware acceleration has become a major topic for efficient rendering of large amounts of glyphs for high-resolution displays. Starting from the ray tracing of spheres and ellipsoids [4] where an analytical projection is possible, sphere tracing and ray tracing became important for superquadrics (e.g., Sigg et al. [18, 7]) where no analytical intersection can be calculated. Hlawitschka et al. [5] presented a method of rendering superquadrics on the GPU-based evaluation of the glyph's gradient function along the ray of sight and using a gradient descent method to approach the surface. In both cases, heuristics have been used to discard unused fragments early on to speed up calculations. Both methods fail in terms of performance for more complex surface functions.

Only recently, Peeters et al. [15] were the first to publish a method to display fourth-order glyphs in spherical harmonic representation using hardware-accelerated ray tracing for spherical harmonic functions given by

$$\Phi(\theta, \varphi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} a_l^m Y_l^m(\theta, \varphi),$$

where θ and ϕ are the polar and the azimuthal angle, respectively, Y_l^m is the spherical harmonic function of degree l and order m, and a_l^m are the factors defining the function. After a bounding-sphere test for early ray termination, the ray is sampled at a constant step size using a sign test on an implicit function derived from Eq. 2 to check whether the surface is hit. If a possible intersection is found, a binary search refines the intersection up to a visually reasonable level.

Several publications describe rendering implicit surfaces on the GPU [8, 9], but most of them are not suitable because the simplicity of Peeter et al.'s approach simply outperforms the "optimizations" suitable for more complex settings.

Our method presented here differs from the method by Peeters et al. in various ways. First, we compute all values using the Cartesian tensor representation to avoid the use of trigonometric functions. Second, our method is not limited to symmetric glyphs of order four, but can be used for a wider range of glyphs, especially glyphs of higher order. Third, we present a method that automatically optimizes the code to the given tensor representation and, therefore, the method is optimal in memory requirement and necessary computations for lower-order glyphs as well as for

higher-order glyphs. Finally, we tested our method on higher-order glyphs to ensure its suitability for data representations such as those required for the spherical deconvolution method by Tournier et al. [20].

3 Method

We first derive a function representation suitable for rendering higher-order tensor data. We then show how to optimize the rendering and, finally, introduce a different approach using high-quality ray casting.

3.1 Implicit Function Representation

Spherical harmonics basis representations have proven to be a standard method of computing and storing derived data from medical images [6, 3, 17, 16]. In general, the explicit, parameterized representation of the surface described in spherical coordinates is

$$f : S^2 \to \mathrm{R}^3$$

$$f(\theta, \varphi) = v(\theta, \varphi) \sum_{l,m} a_l^m Y_l^m(\theta, \varphi),$$

where $v(\theta, \phi)$ denotes a normalized vector pointing in direction (θ, ϕ), and Y_l^m is called the spherical homogeneous polynomial of degree l and order m. Let θ and ϕ be the latitudinal and longitudinal angles indicating a point $\mathbf{p} \in \mathrm{R}^3$. The function can be written as an implicit function with the variable $\mathbf{p}$

$$v(\theta, \varphi) \sum_{l,m} a_l^m Y_l^m(\theta, \varphi) - p = \mathbf{0}$$

or simply

$$\sum_{l,m} a_l^m Y_l^m(\theta, \varphi) - \|\mathbf{p}\| = 0.$$

Nevertheless, a transform from spherical coordinates to Cartesian coordinates seems appropriate to avoid trigonometric functions and the evaluation of Legendre polynomials, which both are numerically unstable at the poles and tend to be computationally challenging. Given data in spherical harmonic coordinates described by the linearized version of the weighting factors $\mathbf{w}_i$, the spherical harmonic basis VY_i, and the tensor $\mathbf{T}_i$, the matrix given by

$$M : m_{ij} = \frac{\langle Y_i(S^2), T_j(S^2) \rangle_{S^2}}{\|T_j\|_{S^2}}$$

defines the transform from spherical harmonic space to tensor space [14, 2]. The expression $\langle \cdot, \cdot \rangle_{S^2}$ denotes the scalar product on the sphere and the linearized tensor after the transform is

$$\mathbf{t} = M\mathbf{w}.$$

A change of coordinate system leads to a representation in harmonic polynomials that can be written using an n-th order tensor $T^{(n)}$ as

$$f_n(\mathbf{x}) = \mathbf{x} T_{i_1 i_2 i_3 \ldots i_n} \mathbf{x}_{i_1} \mathbf{x}_{i_3} \mathbf{x}_{i_3} \cdots \mathbf{x}_{i_n}. \tag{3}$$

When evaluating f at an arbitrary point $\mathbf{p}$, an implicit function representation for $\|\mathbf{p}\| \neq 0$ is derived from

$$\|f_n(\mathbf{p})\| - \|\mathbf{p}\|^{n+1} = 0,$$

which takes into account that $f(\mathbf{p})$ is a polynomial of order $n+1$ regarding the radial directions (i.e., regarding $r = \|\mathbf{p}\|$). [1]

3.2 Surface Normal

Whereas the normal in glyph-based techniques is usually estimated using finite differences to determine the tangent space and calculate this tangent space's normal, using the previous definition of the spatial function, we can compute the glyph's normal implicitly. The normal is given by the gradient at points on the isosurface (contour), i.e.,

$$\frac{\partial f(\mathbf{p}) - \|\mathbf{p}\|}{\partial p_i} = \frac{\partial f(\mathbf{p})}{\partial p_i} - \frac{\partial \|\mathbf{p}\|}{\partial p_i}$$
$$= \frac{\partial f(\mathbf{p})}{\partial p_i} - 2p_i \|\mathbf{p}\|$$

---

[1] We can rewrite f_n to a function f'_n equivalent to the spherical harmonics case by scaling the function by its distance from the center

$$f'_n(\mathbf{p}) = \frac{\mathbf{p}}{\|\mathbf{p}\|} \frac{f_n(\mathbf{p})}{\|\mathbf{p}\|^n}$$

and using the implicit function

$$f'_n(\mathbf{p}) - \|\mathbf{p}\| = 0$$

we get

$$\frac{\mathbf{p}}{\|\mathbf{p}\|} \frac{f_n(\mathbf{p})}{\|\mathbf{p}\|^n} - \|\mathbf{p}\| = 0$$
$$\frac{1}{\|\mathbf{p}\|} \frac{f_n(\mathbf{p})}{\|\mathbf{p}\|^n} - 1 = 0$$
$$f_n(p) - \|\mathbf{p}\|^n = 0$$

4 Implementation

We use hardware-accelerated ray tracing for rendering the implicit function given in Eq. 3. As the approach of Peeters et al. [15] targets the same class of functions, we closely relate their approach that consists of 1) copying the data to the vertex shader; 2) utilizing early ray termination using a bounding sphere; 3) approximation of intersections of the ray with the implicit surface by sampling the path using a fixed step size and doing a sign check of the implicit function; 4) refining the point by a bisection algorithm; and, finally 5) computing the surface normal for lighting. In the following sections, we only point out the major differences of the two approaches and refer the reader to the original paper for technical details.

Setup. We transform the ray into a local coordinate system so that the glyph lies in $[-1,1]^3$ and its center lies at the origin. This simplifies the ray-glyph intersection computation as all calculations are performed relative to the glyph's center.

Copying Data to the GPU. As we are dealing with higher-order tensors, the number of scalars representing a single tensor value is large and, starting with the 28 values of a symmetric sixth-order tensor, reaches the limit current desktop hardware allows us to pass to the shaders and between the shaders (typically 24 floating-point variables). To bypass this issue, we have to store all data in texture memory and access the data independently for each pixel *fragment shader*. An example of the memory layout we use is shown in Figure 3 using a symmetric fourth-order tensor. The exact location of the data is given by the tensor's index, which can be stored in any free vertex attribute. Obviously, using texture memory and the need to extract the tensor values per fragment affects the speed of the algorithm, but we keep the number of texture look-ups small (four look-ups for order four, seven look-ups per pixel for order six, $\lceil (n+1)(n+1)/8 \rceil$ look-ups for order n). In addition, the texture look-ups are not required for the early ray termination step and, therefore, all texture look-ups are performed after this step.

Ray–Surface Intersection. The preliminary ray–glyph intersection step is the most important step in the algorithm. A failure to detect an intersection here will discard the fragment and this error cannot be corrected later on. Therefore, the sampling step size has to be sufficiently small to ensure that all rays that hit the glyph are correctly detected and, in addition, that they intersect with the right "lobe" of the glyph. If they accidentally miss a part of the glyph, sampling artifacts occur.

number of steps	bisect 0	bisect 1	bisect 2	bisect 3	interpolation
0.01	5.0	5.0	5.0	5.0	4.8
0.02	8.6	9.3	9.1	9.0	8.5
0.05	18.6	19.0	18.7	18.0	18.0

Table 1 Comparison of the performance for different rendering modes using frames per seconds. The same data set is rendered repeatedly under the same viewing angle for about two seconds and the average frame rate is shown here.
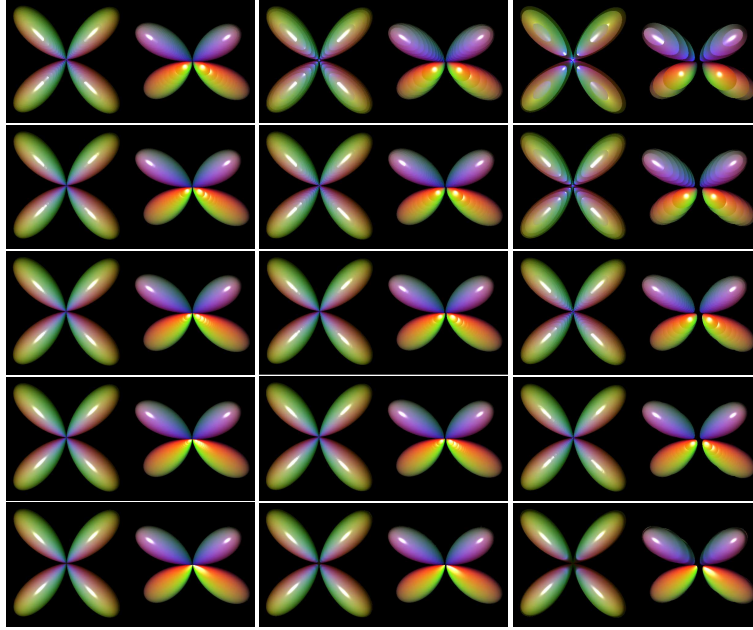
Fig. 1 Rows from top to bottom: comparison of different rendering modes using fixed step size; fixed step size combined with one, two, and three bisection steps; and fixed step size combined with interpolation, respectively. Columns from left to right: two glyphs rendered using a step size of 0.01, 0.02, and 0.05, respectively. None of the methods can improve the quality of the shape in those parts that are not captured by the initial sampling. Those artifacts become especially visible in the center of the right glyph, which has the same shape as the left one but is slightly tilted which makes it numerically challenging. When the glyph is correctly sampled, a linear interpolation performs better than two bisection steps. Depending on the application and the size of the glyphs, a step size of 0.2 and linear interpolation as shown in the center bottom panel seems to provide the best tradeoff: A coarser initial sampling is crucial especially for rays that do not hit the glyph, as those account for most function evaluations and the linear interpolation does not need any additional function evaluations but provides visually better results than three bisection steps.

Refinement Step. Given a proper pair of points, one lying outside the glyph towards the eye point and one lying inside the glyph, bisection is in fact an efficient method to increase the quality as it requires a single function evaluation to reduce the interval by a factor two, giving a binary digit in precision. In our experiments we found that a final linear interpolation of the given interval improves the quality of the rendering tremendously *without* requiring an additional interpolation because we store the previous pair of function values and the size of the search interval. Let f^a, f^b be the function values for parameter a and b, respectively, we compute the ray parameter t

$$t = \frac{-f^a}{f^b - f^a}(b - a) + a.$$

This reduces stair-stepping artifacts when using lower sampling rates along the ray as seen in Figure 1.

Overcoming the Singularity at the Origin. Similarly to the spherical harmonics definition, the implicit function defined here has high-frequency components close to the glyph's center, i.e., the origin of our local coordinate system. Whereas moving the coordinate system to the glyph's center in general ensures higher numerical precision for some operations, it does not solve the problem of under-sampling by the simple ray casting. Even though the glyphs shown in Figure 1 are rare in standard imaging techniques like q-Ball imaging [21, 22], in the post-processing step and to ensure better visibility, a sharpening filter is applied to the surface function that introduces high-frequency components and, in general, removes the major part of the isotropic behavior. Therefore, it is not uncommon to have glyphs that touch their center point. There are two ways to overcome this problem: First, one can use a redefinition of the glyph that avoids those situations and, second, one could use a finer sampling of the data. Even though the first approach changes the glyph, it can be used to visualize the data since the user is aware of this fact.

As a finer sampling of the data reduces the speed of the algorithm tremendously (which is mainly due to the fact that a large number of rays never hit the surface and, therefore, account for the majority of function evaluations), we adapt the sampling to the expected frequency pattern of the surface by changing the step size to a finer step size in closer vicinity to the center of the glyph while maintaining the coarser step size at the outer parts. A result of this approach is shown in Figure 2.
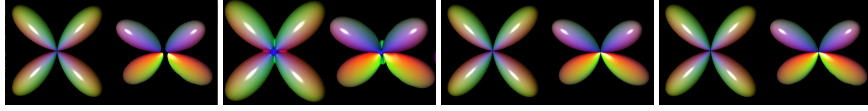

Fig. 2 The problem of under-sampling the glyph in the center (first picture) can be solved in different ways. The introduction of a basic isotropic component to every glyph avoids high-frequency components in the implicit function and, therefore, produces a picture without gaps while only slightly changing the glyph (second picture). A better solution is an approach based on adaptive sampling towards the center of the glyph. The two pictures are rendered with a step size of 0.2 and 0.5 (third and fourth picture, respectively), and increase the sampling step size to a third of their original step size towards the center. While slowing down rendering speed slightly (from 18 fps to 15 fps in our test case), we are able to produce more precise visualizations.

Function Implementation. Depending on the type of input data, we automatically generate the suitable rendering code. This is useful as the tensor function simplifies for lower-order tensors as well as for symmetric tensors. Especially the reduced amount of texture look-ups during rendering leads to an increase in performance.

We compute the function depending on the data using

$$f(x,y,z) = \sum_i \left[x^{h_{i,x}} y^{h_{i,y}} z^{h_{i,z}} \right] - \sqrt{x^2 + y^2 + z^2}^{n+1} \tag{4}$$

t^0_{0000}	t^0_{0001}	t^0_{0002}	t^0_{0011}	t^0_{0012}	t^0_{0022}		t^0_{2222}
t^1_{0000}	t^1_{0001}	t^1_{0002}	t^1_{0011}	t^1_{0012}	t^1_{0022}		t^1_{2222}
t^2_{0000}	t^2_{0001}	t^2_{0002}	t^2_{0011}	t^2_{0012}	t^2_{0022}		t^2_{2222}
t^3_{0000}	t^3_{0001}	t^3_{0002}	t^3_{0011}	t^3_{0012}	t^3_{0022}		t^3_{2222}
t^{N-1}_{0000}	t^{N-1}_{0001}	t^{N-1}_{0002}	t^{N-1}_{0011}	t^{N-1}_{0012}	t^{N-1}_{0022}		t^{N-1}_{2222}

Fig. 3 Layout of the tensor in texture memory using the example of a symmetric fourth-order tensor: Depending on the graphics board used, the use of 1D or 2D textures may result in better performance. Usually, the tensor data is not a power of two and, therefore, leaving memory elements empty may be necessary to achieve better frame rates. Those texture elements can be used as additional information for example for color coding or scaling.

for non-symmetric tensors. The optimized version for symmetric tensors is

$$f(x,y,z) = \sum_i \left[\pi_i x^{h_{i,x}} y^{h_{i,y}} z^{h_{i,z}} \right] - \sqrt{x^2 + y^2 + z^2}^{n+1} \tag{5}$$

where

$$\pi_i = \frac{3^n}{h_{i,x}! h_{i,y}! h_{i,z}!}$$

is the number of occurrences of each term and $h_{i,x}$ is the number of occurrences of the digit 0 (1, 2) in the number i represented to the base 3, i.e., the power of x (y, z), respectively.

For higher shading quality and faster evaluation of the gradient, we also provide the partial derivatives of the function used as surface normal, which can be derived analytically from Equation 5, e.g., for the symmetric case

$$\frac{\partial f(x,y,z)}{\partial x} = \sum_i \left[v \pi_i h_{i,x} x^{h'_{i,x}} y^{h_{i,y}} z^{h_{i,z}} \right] - 2x(n+1)\sqrt{x^2 + y^2 + z^2}^n,$$

$$\frac{\partial f(x,y,z)}{\partial y} = \sum_i \left[v \pi_i h_{i,y} x^{h_{i,x}} y^{h'_{i,y}} z^{h_{i,z}} \right] - 2y(n+1)\sqrt{x^2 + y^2 + z^2}^n, \text{ and}$$

$$\frac{\partial f(x,y,z)}{\partial z} = \sum_i \left[v \pi_i h_{i,z} x^{h_{i,x}} y^{h_{i,y}} z^{h'_{i,z}} \right] - 2z(n+1)\sqrt{x^2 + y^2 + z^2}^n,$$

with

$$\begin{cases} h'_{i,a} = h_{i,a} - 1 & \text{and } v = 1 \quad \forall h_{i,a} > 0, \\ h'_{i,a} = 0 & \text{and } v = 0 \quad \text{otherwise.} \end{cases}$$

Given the flexibility of automatic glyph function generation, an implementation of the spherical harmonics case described by Peeters et al. [15] is possible with only minor changes in our existing code. We used the real-valued definition as defined

by Descoteaux et al. [3] using sine and cosine of θ and φ, and the radius r as input parameters of a modified spherical harmonic function. Again, we store the parameter vector a linearly in a texture similarly to the tensor parameter texture discussed before.

Optimization. The obvious drawback of this automatic code generation is the missing manual optimization of code. There are many calculations of powers that are used in several places across the functions that could be reused. This is even partially true across functions; the function evaluation itself shares major parts with the gradient evaluation. For three reasons, we did not explore this additional potential: When looking at the generated compiled code, many of these optimizations already were performed by the compiler and we doubt there is much room for optimization at this point. Second, most of these optimizations require additional storage which is limited on the graphics board and, therefore, may limit the number of threads running on the GPU in parallel. Third, besides the additional memory requirement, cross-function storage between the function evaluation and the gradient vector calculation is not possible when interpolating the parameter to increase precision (as done in the bisection and in our linear interpolation approach).

Rendering. We implemented our approach using Nvidia's Cg and OpenGL in our existing visualization system. We trigger the fragment shader by rendering the front-facing quadrilaterals of a bounding box around the glyph, which is projected to image space in the vertex shader and parameterized using local coordinates that help with the calculation the ray parameters. Even though we could add another early ray termination step here by ignoring all pixels that do not lie within the projection of the bounding sphere, we currently skip this step for simplicity. The sphere intersection test is responsible for discarding about $\frac{\pi}{6}$ of the rays hitting the bounding box [12] and, therefore, is a mandatory optimization. Based on the ray–sphere intersection, the ray is parameterized and the main algorithm starts as described in the preceding paragraphs.

Color Coding. Whereas there are many different ways to color-code second-order tensor information, to the best of our knowledge, only two major color-coding schemes are used for higher-order tensors: The frequently used color coding by scalar value and the less frequently used color coding by direction. Both schemes fit seamlessly in our approach as the information required can be obtained from the surface position itself. Given a point $\mathbf{p}$ on the surface and the normalized vector $\bar{\mathbf{p}}$ pointing from the origin towards $\mathbf{p}$,

$$c = \mathsf{colormap}(\|\mathbf{p}\|)$$

represents the first and

$$c = \mathsf{RGB}(\mathrm{abs}(\bar{p}_x), \mathrm{abs}(\bar{p}_y), \mathrm{abs}(\bar{p}_z))$$

the latter color coding scheme. Both can be implemented efficiently in the fragment shader, as the only required information, the hit point $\mathbf{p}$, is already computed. Examples of different color codings are shown in Figure 5.

5 Robust intersection using interval and affine arithmetic

Even though the constant step size approach has proven to be efficient and we have shown how to improve the rendering quality even further, we introduce another glyph rendering approach, relying on interval analysis, for even more accurate results. This approach is based on Knoll et al.'s implicit surface rendering [8, 9], which can be used to render almost any implicit surface up to a user-defined precision. The algorithm is based on interval arithmetic (IA) and reduced affine arithmetic (RAA) which are sketched below. We integrated Knoll's algorithm in our software and may use it as an option for the glyph rendering.

Interval arithmetic and reduced affine arithmetic. We first implemented an IA and an RAA library in Cg and re-implemented the glyphs functions using these two new types. Interval arithmetic was introduced by Moore [10] as an approach to bounding numerical rounding errors in floating point computation. IA is particularly well-known for its robust rejection test, especially for ray tracing, but it can suffer from overestimation problems. To address this issue, a few decades later, affine arithmetic (AA) was developed by Comba & Stolfi [1]. In practice we might want to truncate the number of elements composing an affine form due to memory consumption, in which case it is referred to as reduced affine arithmetic. Intuitively, if IA approximates the convex hull of a function f with a bounding box, AA employs a piecewise first-order bounding polygon, such as the parallelogram in Fig. 4. For our class of glyph functions, RAA is up to five times faster than IA.
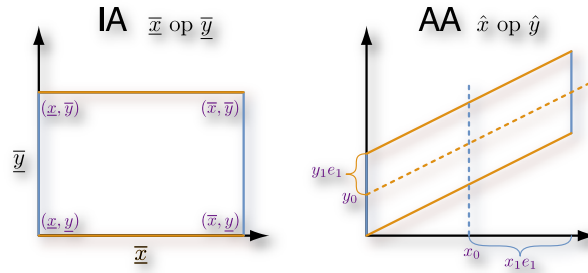

Fig. 4 Bounding forms resulting from the combination of two interval (left) and affine (right) quantities.

Rejection test. Moore's fundamental theorem of interval arithmetic [10] states that for any function f defined by an arithmetical expression, the corresponding interval evaluation function F is an *inclusion function* of f: $F(\overline{x}) \supseteq f(\overline{x}) = \{f(x) \mid x \in \overline{x}\}$ where $\overline{x}$ is an interval. Given an implicit function f and a n-dimensional bounding box B defined as a product of n intervals, we have a very simple and reliable rejection test for the box B not intersecting the image of the function f (in

our case, the glyph surface), $0 \notin F(B) \Rightarrow 0 \notin f(B)$. This property can be used in ray tracing or mesh extraction for identifying and skipping empty regions of space.

Comparison with our previous glyph rendering algorithm. IA/AA numerical approaches are more robust than point-sampling methods (e.g. using the rule of signs) but they require more computations and therefore are slower; they work well when a function is generally non-Lipschitz (Knoll et al. [9]), as is the case at singularities in our functions. However, our glyph functions are generally Lipschitz outside these singularities, causing point-sampling approaches to work well in practice. Note that brute-force methods like point-sampling work best on the GPU [19]. When combined with a bisection scheme (with enough iterations) on the GPU, IA-based approaches provide high-quality glyphs at interactive rates [9] regardless singularities such as the one at the center; in this way IA helps solving the singularity problems discussed previously.

6 Results

We applied our algorithm to several real-world data sets. The first data set shown in Figure 5 is a $3 \times 3 \times 3 \times 3$ stiffness tensor of order four generated by Alisa Neeman [11].

The second data set is a human brain image data set of a healthy volunteer and was provided by the Max Planck Institute for Human Cognitive and Brain Sciences, Leipzig, Germany. It had been acquired using 60 gradient directions, using three-times averaging and 19 b0 images on a three Tesla Siemens Trio scanner. The input was mapped to symmetric tensors using least-squares fitting [6]. It turns out that the standard resolution glyphs are shown is quite small and, therefore, a lower-quality rendering can be employed without notable difference in quality of the final visualization.

Performance. We tested our algorithm on a late 2008 Apple MacBook Pro 15" laptop computer with the build-in Nvidia GeForce 9600M GT graphics board and 512MB of VRAM. There, using an OpenGL window size of 800x600 pixel, we achieve a rendering typical speed of 15 to 20 frames per second. Even though this is lower than the speed of standard tessellation-based approaches, there is almost no overhead when pre-processing the geometry. Therefore, changing the glyph's location, e.g., by changing the plane in the data set or modifying a region of interest, can be done at almost the same frame rates.

Memory Requirements. The memory use highly depends on the amount of glyphs displayed, whether display lists are generated or not, and the order of the tensor information. The texture memory used has the same size as the data in main memory. Only when an additional padding is used, e.g., when power-of-two textures lead to increased performance, slightly more memory is used. Additional memory on the GPU may be required to store the bounding box information, which could be generated on the fly using geometry shaders.
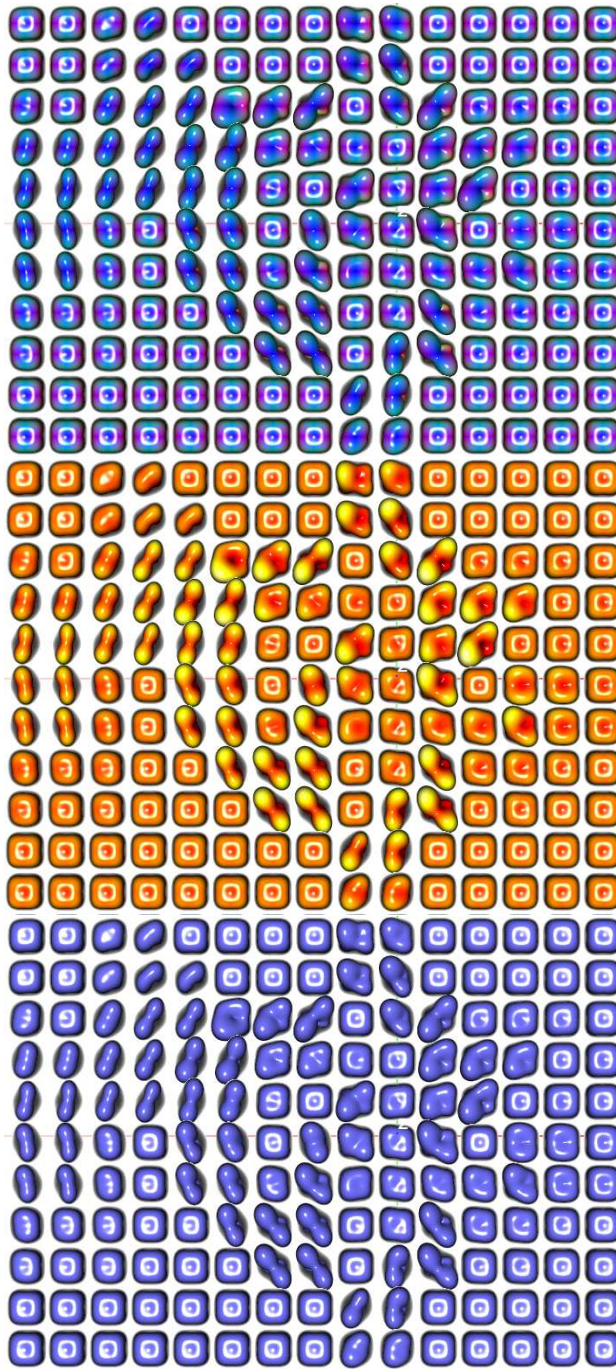
Fig. 5 A fourth-order stiffness tensor data set from material sciences shown using RGB color coding, a magnitude color coding ranging from black to red to yellow, and a uniform blue color coding. The data shows a simulated force applied to a brick of complex material.

System	#glyphs	tensor	#triangles	time generation	fps rendering
L	258	4n	(res 7)	1s	$60s^{-1}$
L	258	4n	(res 10)	3s	$59s^{-1}$
L	258	4n	(res 20)	9s	$50s^{-1}$
L	258	4n	ray tracing	<1s	$18s^{-1}$

Table 2 Number of glyphs, type of glyph (order and s=symmetric n=non-symmetric), number of triangles per glyph, time to create geometry, and frame rate in frames per second (fps) of final rendering. The computer system is a MacBook Pro 15" Laptop Computer.

7 Discussion

Due to the flexibility of the implicit function ray tracing, the presented approach can be used to display different kinds of glyphs: Using symmetric, positive-definite second-order tensors as input, the output is the Reynold's glyph [11, 6]. Using the function

$$f(\mathbf{p}) = \frac{1}{p_x^2}T_{xx} + \frac{2}{p_xp_y}T_{xy} + \frac{2}{p_xp_z}T_{xz} + \frac{1}{p_y^2}T_{yy} + \frac{2}{p_yp_z}T_{yz} + \frac{1}{p_z^2}T_{zz} = c$$

leads to a representation of the ellipsoidal glyph, which can be selected as an option in our implementation. Using the functions provided by Özarslan and Mareci [13] we could even render this glyph directly from higher-order data. However, as there is an explicit ray-ellipsoid intersection algorithm, this is not recommended. Even though the superquadric tensor glyph [7] can be represented by implicit functions, currently, it does not fit in this scheme as the function is based on the relation of two parameters (cf. [7] for details.) Even though this could be implemented as well, we advice to use more efficient implementations as presented by Hlawitschka et al. [5], for example.

In contrast to displaying geometry, which is usually copied to the GPU in smaller packets, our method relies on most data residing in GPU memory. This implies that the GPU's main memory and the order of the glyph limits the number of glyphs that can be displayed in one rendering pass. This can be circumvented by splitting the data set into smaller subsets that are rendered successively.

Future research will target optimizations to reduce the number of glyphs that are rendered even though they are occluded by other glyphs. Deferred shading is not suitable in our case because most of the time is used for the actual ray-glyph intersection, which has do be done anyway. In addition, deferred shading requires additional data lookups and, when a small number of glyphs is occluded, this would slow down the system tremendously.

8 Acknowledgements

References

1. Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: Proc. VI Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'93), pp. 9–18 (1993). URL citeseer.ist.psu.edu/dihlcomba93affine.html
2. Descoteaux, M., Angelino, E., Fitzgibbons, S., Deriche, R.: Apparent diffusion coefficients from high angular resolution diffusion imaging: Estimation and applications. Magnetic Resonance in Medicine **56**, 395–410 (2006)
3. Descoteaux, M., Angelino, E., Fitzgibbons, S., Deriche, R.: Regularized, fast, and robust analytical q-ball imaging. Magnetic Resonance in Medicine **58**, 497–510 (2007)
4. Gumhold, S.: Splatting illuminated ellipsoids with depth correction. In: VMV, pp. 245–252 (2003)
5. Hlawitschka, M., Eichelbaum, S., Scheuermann, G.: Fast and memory efficient GPU-based rendering of tensor data. In: Proceedings of the IADIS International Conference on Computer Graphics and Visualization 2008 (2008)
6. Hlawitschka, M., Scheuermann, G.: HOT-lines — tracking lines in higher order tensor fields. In: C.T. Silva, E. Gröller, H. Rushmeier (eds.) Proceedings of IEEE Visualization 2005, pp. 27–34 (2005)
7. Kindlmann, G.: Superquadric tensor glyph. Joint EUROGRAPHICS – IEEE TCVG Symposium on Visualization (2004)
8. Knoll, A., Hijazi, Y., Hansen, C., Wald, I., Hagen, H.: Interactive ray tracing of arbitrary implicits with simd interval arithmetic. Proceedings of the 2nd IEEE/EG Symposium on Interactive Ray Tracing pp. 11–18 (2007)
9. Knoll, A., Hijazi, Y., Kensler, A., Schott, M., Hansen, C., Hagen, H.: Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. Computer Graphics Forum 2008 **28**(1), 26–40 (2008)
10. Moore, R.E.: Interval Analysis. Prentice Hall, Englewood Cliffs, NJ (1966)
11. Neeman, A.: Visualization techniques for computational mechanics. Dissertation, University of California, Santa Cruz (2009). URL http://users.soe.ucsc.edu/ aneeman/
12. Nienhuys, H.W.: Ray tracing news (1997)
13. Özarslan, E., Mareci, T.H.: Generalized diffusion tensor imaging and analytical relationships between diffusion tensor imaging and high angular resolution diffusion imaging. Magnetic Resonance in Medicine **50**, 955–965 (2003)
14. Özarslan, E., Vemuri, B.C., Mareci, T.H.: Higher rank tensors in diffusion MRI. In: J. Weickert, H. Hagen (eds.) Visualization and Processing of Tensor Fields, pp. 177–187. Springer–Verlag Berlin Heidelberg (2006)
15. Peeters, T., Prčkovska, V., van Almsick, M., Vilanova, A., ter Haar, R.: Fast and sleek glyph rendering for interactive hardi data exploration. In: Pacific Visualization (2009)
16. Schultz, T.: Feature extraction for visual analysis of DW-MRI data. Ph.D. thesis, Universität des Saarlandes (2009)

17. Schultz, T., Seidel, H.P.: Estimating crossing fibers: A tensor decomposition approach. IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Visualization) **14**(6), 1635–1642 (2008)
18. Sigg, C., Weyrich, T., Botsch, M., Gross, M.: GPU–based ray–casting of quadric surfaces. Eurographics Symposium on Point–Based Graphics (2006)
19. Singh, J.M., Narayanan, P.: Real-time ray tracing of implicit surfaces on the gpu. IEEE Transactions on Visualization and Computer Graphics **99**(2). DOI http://doi.ieeecomputersociety.org/10.1109/TVCG.2009.41
20. Tournier, J.D., Calamante, F., Connelly, A.: Robust determination of the fibre orientation distribution in diffusion mri: Non-negativity constrained super-resolved spherical deconvolution. NeuroImage **35**, 1459–1472 (2007)
21. Tuch, D.S.: Q-ball imaging. Magnetic Resonance in Medicine pp. 1358–1372 (2004). DOI 10.1002/mrm.20279
22. Tuch, D.S., Wisco, J.J., Khachaturian, M.H., Ekstrom, L.B., Kötter, R., Vanduffel, W.: Q-ball imaging of macaque white matter architecture. Philosophical Transactions of The Royal Society B (2005)