Coherency-Based Curve Compression for High-Order Finite Element Model Visualization

Alexander Bock, Erik Sundén, Bingchen Liu,
Burkhard Wünsche, *Member, IEEE*, and Timo Ropinski, *Member, IEEE*

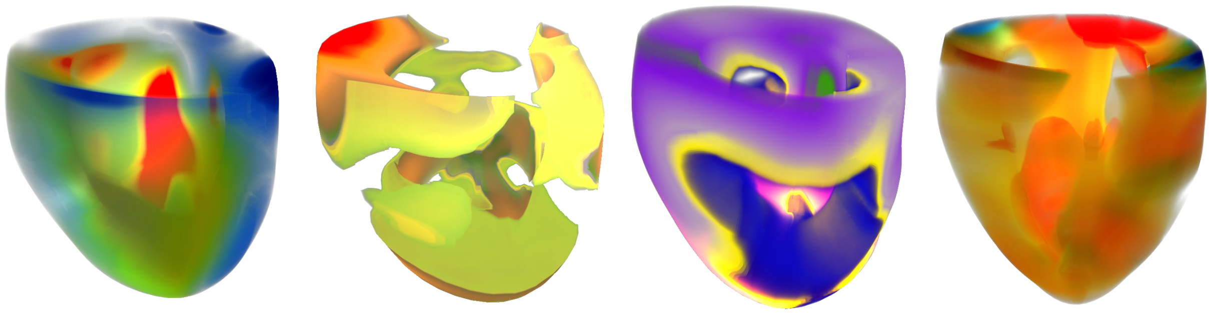


Fig. 1. Application of our visualization approach to a multi-parametric FE model of the left ventricle of a human heart. The images show volumetric renderings of different strain directions defined over the FE model: radial strain, circumferential strain, longitudinal strain as well as the three former combined (*left to right*). The presented approach allows for interactive data exploration by changing the transfer function and other relevant rendering parameters.

Abstract—Finite element (FE) models are frequently used in engineering and life sciences within time-consuming simulations. In contrast with the regular grid structure facilitated by volumetric data sets, as used in medicine or geosciences, FE models are defined over a non-uniform grid. Elements can have curved faces and their interior can be defined through high-order basis functions, which pose additional challenges when visualizing these models. During ray-casting, the uniformly distributed sample points along each viewing ray must be transformed into the material space defined within each element. The computational complexity of this transformation makes a straightforward approach inadequate for interactive data exploration. In this paper, we introduce a novel coherency-based method which supports the interactive exploration of FE models by decoupling the expensive world-to-material space transformation from the rendering stage, thereby allowing it to be performed within a precomputation stage. Therefore, our approach computes view-independent proxy rays in material space, which are clustered to facilitate data reduction. During rendering, these proxy rays are accessed, and it becomes possible to visually analyze high-order FE models at interactive frame rates, even when they are time-varying or consist of multiple modalities. Within this paper, we provide the necessary background about the FE data, describe our decoupling method, and introduce our interactive rendering algorithm. Furthermore, we provide visual results and analyze the error introduced by the presented approach.

Index Terms—Finite element visualization, GPU-based ray-casting.

1 Introduction

The finite element (FE) method is a computational technique frequently applied in science, medicine, and engineering to solve partial differential and integral equations [39]. The technique enables the discretization of complicated domains into arbitrarily shaped elements, in which material properties can be defined using nodal parameters together with potentially high-order interpolation functions. Thus, it becomes possible to incorporate material properties at every point rather than a limited number of sample points. Applications range from simulations of physical processes over derivation of new entities from measured data, for example, computation of material strain from displacement data, to analyses and comparisons of data defined over objects with similar topology. All of these applications involve large amounts of multi-dimensional and multi-parametric data defined over arbitrary shaped grid structures. A prominent application example from bioengineering, in which FE visualization is essential, is the strain analysis of the human heart. An understanding of the full 3D strain tensor, represented by a symmetric matrix at each point, is necessary to identify unusual behavior of the heart muscle's deformation and to detect common patterns indicating developing diseases. Since there is little knowledge available about the exact behavior, interactive exploration plays an important role in the quest for new discoveries. By providing immediate visual feedback through rapid image generation, interactivity enables a rapid response to human input. In our case, it should be possible to interactively change all relevant visualization parameters, for example, parameter mapping and view orientation, such that the latency does not hinder the volumetric exploration process. While interactive surface-based visualizations of FE models are well established, volumetric visualization techniques supporting the interactive analysis of the entire structure are rare [35]. This is largely due to the fact, that accessing the materials defined over high-order basis functions, which needs to be done for each sample in a volumetric visualization approach, involves high computational complexity.

- *Alexander Bock, Erik Sundén and Timo Ropinski are with the Scientific Visualization Group, Linköping University, Sweden, E-mail: {alexander.bock, erik.sunden, timo.ropinski}@liu.se.*
- *Bingchen Liu and Burkhard Wünsche are with the Computer Science Department, University of Auckland, New Zealand, E-mail: {bliu035, b.wuensche}@auckland.ac.nz.*

Manuscript received 31 March 2012; accepted 1 August 2012; posted online 14 October 2012; mailed on 5 October 2012.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

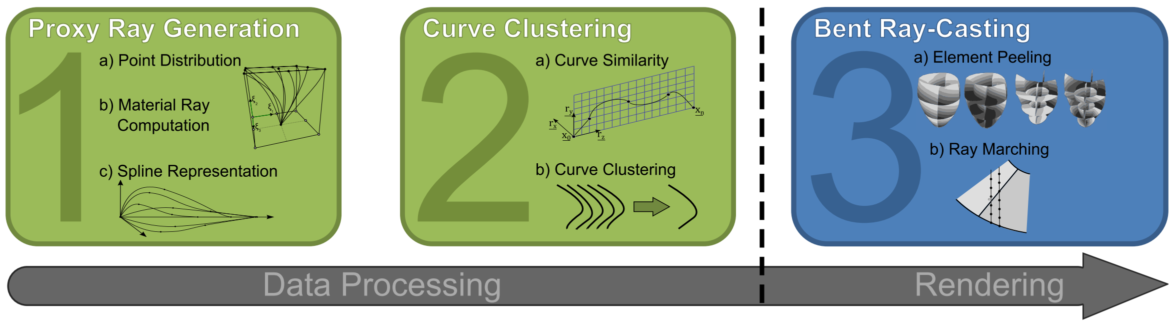


Fig. 2. **Workflow.** To enable interactive ray-casting of curvilinear FE models, we exploit a multi-stage preprocessing approach. In the first step, we compute representative curved rays in material space, which are represented as splines and transformed in the second step into a common coordinate system. To reduce the amount of data for the subsequent rendering stage, we apply a rotation invariant curve clustering in the third step. The resulting data is then passed to the GPU-based rendering stage, where ray marching is performed.

In this paper, we introduce an interactive volumetric ray-casting algorithm for FE models that exploits ray coherency. When ray-casting FE models, viewing rays are straight lines in world space, but bent to curves in material (or ξ) space. Consequently, each sample point on a straight viewing ray must be transformed during rendering onto the corresponding curve in material space before the FE data can be fetched. To avoid this computationally complex transformation, we exploit ray coherency within a curve clustering approach, and thus precompute a view-independent proxy representation of the curved rays that can be accessed efficiently during rendering. The workflow of our approach is shown in Figure 2. In the first step we compute a representative subset of curved rays in material space, which are represented by splines. This subset is then reduced through clustering within the second curve compression step to obtain a lower number of curves, to which we refer as *proxy rays*. Assuming that the proxy rays sufficiently represent the coherent curves of a FE model, we can access them in the third step, the rendering step, where we can omit the expensive world-to-material space transformation. This allows us to enable volumetric FE model visualization of potentially multi-parametric FE models on modern graphics processing units (GPUs) at interactive frame rates (see Figure 1). As we achieve this interactivity by exploiting preprocessing algorithms and data structures accessed during rendering. The costs of our approach are increased precomputation times as well as memory consumption, which are discussed in more detail in Sections 7.3 and 7.4. While our current implementation is constrained to hexahedral FE models, we will discuss the implications for other cell types within Section 6.

2 Related Work

Interactive high-quality rendering of FE models, incorporating the interior of the elements, is a demanding task. Often, the elements are resampled to lie on a common regular uniform grid [35]. While this enables a direct visualization using standard volume rendering algorithms, it also results in sampling-based errors as well as a loss of the easy access to element boundaries. As a consequence, many researchers have investigated rendering techniques that enable a more direct FE visualization by maintaining the relationship between the world and the material coordinate system.

Wünsche introduced an isosurface-based approach for the visualization of biomedical tensor field data [37], which has been applied to strain analysis. Another technique for rendering isosurfaces was presented by Nelson et al. in which they considered spectral high-order FE meshes and introduced the idea of an error budget [26]. Schroeder et at. developed a framework for isosurface rendering by tessellating the basis functions into simpler forms which improves the mesh quality [30]. Meyer et al. chose an approach based on particle systems to render isosurfaces efficiently [23].

An alternative to the isosurface-based method was presented by Brasher et al. [4] and Nelson et al. [25], in which cut planes through the high-order finite element data are evaluated. Rose and Ertl proposed a texture-based mesh reduction approach in order to render surface representations of large FE models at interactive frame rates [29]. Other authors have proposed algorithms which exploit a conversion into a tetrahedral mesh representation [28, 20, 10] which, in comparison with regular grid sampling, allow for adaptivity.

To improve rendering quality ray-casting approaches have been proposed. Garrity was one of the first to introduce ray-casting for rendering of unstructured meshes [8]. The technique is suitable for FE models as it tracks numerous entry and exit points of a ray from the cells which are stored in a hierarchical data structure. Hong and Kaufman incorporated the projection approach into ray-casting to speed up the ray traversal and interpolation [14]. They later improved their method by employing a projection of the elements' faces onto the image plane, which allowed them to compute ray intersection points for the ray traversal more efficiently [15]. A similar strategy for finding intersection points was used by Farias et al., who applied a space sweeping approach [7]. Grimm et al. performed the intersection point computation on the GPU [11]. Their hybrid CPU/GPU approach performs a layer peeling on the GPU while the actual ray-traversal is performed on the CPU. Martin et al. utilize a special representation of curvilinear space for transformation between physical space and computational space using a linear approximation [21].

Moreland and Angel presented a method based on a partial pre-integration of the volume rendering integral and achieved interactive frame rates when rendering linear tetrahedral meshes [24]. Wiley et al. demonstrated a technique to perform ray-casting on quadratic curved elements, but did so without regard to interactivity [36]. This method could achieve interactive results on current machines but is limited to quadratic elements, which simplifies the inversion of the necessary coordinate transform.

More recently, Üffinger et al. have presented a distributed visualization approach that allows volumetric visualization of models to be defined through high-order polynomials [34]. As in ours their approach is also based on ray-casting but, in contrast, they circumvent computation of the world-to-material space transformation by transforming the FE solution into a Cartesian reference space with barycentric coordinates. The FE interpolation is then replaced by a compact monomial representation, which the ray-casting kernel can use during sampling in physical space.

3 Curvilinear Finite Element Models

While the concepts presented are, in theory, applicable to any FE model, they unfold their full potential when applied to FE models where the material space is defined through high-order basis functions. The curvilinear element examples presented in this paper employ cubic interpolation functions in each dimension, resulting in an interpolation function that is a degree-9 polynomial. In several domain areas, as for instance in bioengineering [18, 38], the number of FE elements is relatively low and high-order interpolation functions are used.

FE models are defined through two coordinate systems. The *world coordinate system* provides the position and orientation of the elements in space by defining a set of nodal points, $x^{(e_j)}$, for each element

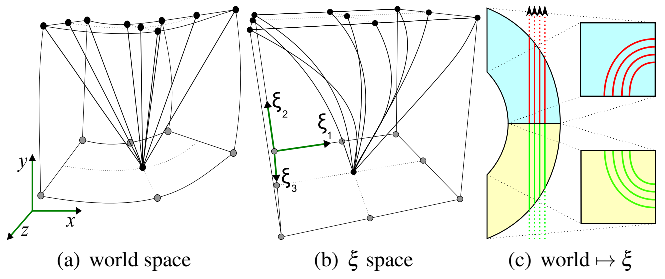


Fig. 3. Transforming straight rays given in world space, *(a)*, to ξ space, *(b)*, causes these rays to be curved. View rays that are parallel and similar in world space, *(c,left)*, will be projected to similar curves in ξ space, *(c,right)*. The dots show the points used for calculating the proxy rays shown, with the dotted lines emphasizing the relevant face.

e_j. These nodal points can be arbitrarily oriented in space. The second coordinate system is the the *material*, or ξ, *coordinate system* which defines elements using simple geometric shapes, such as tetrahedra, prisms, pyramids, or unit cubes, which are decoupled from the actual geometry. The examples in this work are constructed based on unit cubes. In this case, the material coordinate system is cartesian with $\xi_1, \xi_2, \xi_3 \in [0,1]$ as basis for a right-handed coordinate system. World and material coordinate system are connected by requiring each element, e_j, with $\xi_i(1) = x_{2,i}^{(e_j)}$, $i = 1,2,3$ be such that any point in the domain has a world coordinate x with an associated ξ coordinate. This association is provided by a set of mapping functions, $\phi_i(\xi)$. The dependent variables are almost always expressed in ξ coordinates, since ξ coordinates are independent of the element geometry and thus support the analysis of multi-parametric data and the comparison of different FE models. Thus, the dependent variables and the FE interpolation functions are defined in ξ space while the nodal points are defined in world space. This combination allows us to exploit the interpolation functions in order to compute variable values for all world space coordinates lying inside the model. The relevant mathematical concepts for FE modeling used in this paper are described in more detail in [38, 3, 33, 31, 16].

When visualizing such models world coordinates are usually more convenient to use, as they define the position and orientation of the elements. As soon as the dependent variables have to be accessed however, the ξ coordinates need also be taken into account as the dependent variables can only be accessed through transformation into ξ space. Thus, the world coordinate system can be used for finding ray intersection points with the elements, while the actual ray traversal is performed in ξ space. As can be seen in Figure 3, this transformation affects the geometry of the viewing rays, which can end up as curves in ξ space. While the transformation from ξ space to world space can be computed fairly easy as $x(\xi) = \sum_{i=1}^{n} x_i \phi_i(\xi)$, for n nodal coordinates, x_i, and basis functions, ϕ_i, the inverse transformation required during ray-casting is computationally much more demanding. Numerical methods, for example the multi-dimensional Newton method [27], which are often applied to perform this transformation, involve high computational complexity and thus are not feasible in an interactive context. Therefore alternative approaches are necessary to avoid the computational complexity of the world to ξ space transformation in the rendering stage of the visualization pipeline.

4 Coherency-based Curve Compression

In this section we present the basis for a novel ray-casting based algorithm for the interactive exploration of curvilinear FE models.

The *straightforward approach* for ray-casting FE models would be to march along a ray in world space and transform each world space sample coordinate, x, into its corresponding material coordinate, ξ, before accessing the dependent variables. As shown in Table 1, this approach is computationally very demanding and therefore cannot, in general, be performed at interactive frame rates. By using our approach we are able to shift this, originally view-dependent, computation from the rendering stage into the data processing stage of the visualization pipeline. Thus, we can reduce the computational load during rendering and achieve interactive frame rates when volume rendering curvilinear FE models.

To perform the desired shift from the rendering stage into the data processing stage, we exploit the following key observations about the degree of coherency of curvilinear FE models. FE method grids are designed to minimize approximation errors in the solution and to ensure convergence. In particular the Jacobian of the world-to-material coordinate mapping must not contain any singularities, and the grid resolution needs to be high enough that variations of the unknown variables can be approximated adequately with the underlying FE basis functions [17]. This implicit smoothness constraint is exploited in our technique since it also limits the shape variations of neighboring rays in world coordinates mapped into ξ space, i. e., we can assume some degree of intra-element coherency between adjacent rays. Because the Jacobian does not contain singularities we can employ the Newton method, or similar solvers, to compute the inverse of the world-to-material coordinate mapping.

Many FE applications also require some degree of inter-element compatibility, and some degree of symmetry and self-similarity. These characteristics result in a FE mesh with clusters of similar elements, and hence some degree of inter-element coherency. Figure 3 (c) illustrates this coherency as it shows the shape of rays in ξ space (*right*), which adjacently traverse world space (*left*).

In addition, coherency is supported as most models in practical applications need well-behaved elements, i.e. low distortion and an aspect ratio close to one, in order to result in a stable numerical simulation. The approach presented uses precomputation, which is performed within two preprocessing steps as shown in Figure 2. In the first step, we compute a high number of proxy rays in ξ space. These proxy rays act as a view- and resolution-independent spline representations of the actual viewing rays that are cast through the ξ space during rendering. In the second step, we exploit inter- and intra-element coherency to reduce the vast number of proxy rays to a representative subset, which is then used during GPU-based ray-casting in the third step. By exploiting the precomputed data FE models can be explored interactively by changing the transfer function and other rendering parameters. The following two subsections describe the two preprocessing steps, while the GPU-based ray-casting is discussed in Section 5.

4.1 Proxy Ray Generation

To be able to compute a complete set, R, of proxy rays, it is essential that it covers all parts of the FE model. This means that both the world and the ξ space are densely populated with straight and curved rays, respectively. However, since the view direction used during rendering affects the ray traversal direction and the proxy rays are directional, it is not sufficient that all positions within an element are close to a single proxy ray; they must also be close to proxy rays having different di-

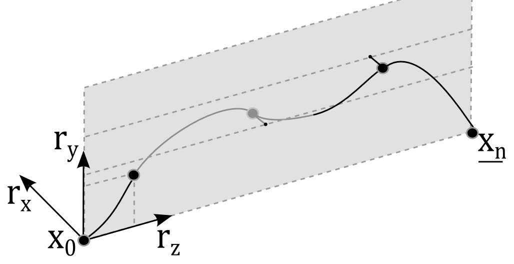


Fig. 4. To achieve a higher degree of similarity during the curve clustering we exploit the orientation invariance of the proxy rays and transform them into a common coordinate system where the principal curve axis coincides with the z-axis. Furthermore we exploit scale invariance to align all exit points.

rections. As the interpolation functions in ξ space can, in principle, be different for each element within one FE model, elements are treated individually during preprocessing and rendering. Finding the desired dense and omnidirectional proxy ray population for an element can be performed in two ways. It can either be done adaptively, by analyzing the actual ray distribution within each element to find areas of high importance and create more rays for these areas, or in a uniform way over each element. As our algorithm is based on the inter- and intra-element coherency the latter distribution is sufficient since we assume a continuity between neighboring rays without high frequency changes.

To obtain such a proxy ray distribution we exploit the intra-element coherency, which is the result of a high similarity between rays in ξ space that enter and exit an element through adjacent coordinates. Therefore we compute a dense and uniform distribution, P, of potential entry and exit points on the surface of each element. While these points should be distributed over the entire surface of an element, the requirements with respect to the uniformity of P are rather loose due to the intra-element coherency. We distribute the potential entry and exit points by performing an equidistant subdivision of all faces of an element. In this step the subdivision parameter, s_f, controls the number of points equidistantly positioned along the u and the v coordinates of a face. Thus the density of P can be directly controlled by changing the subdivision parameter, s_f. To reduce the number of initially generated proxy rays it would also be an option to vary s_f based on the extents of a face in world space. In cases where an element is strongly deformed however, such that one face is much smaller than other faces, this usually indicates strong variations in the FE simulation which makes a higher sample density in world space necessary. Therefore we have decided to choose a constant, but sufficiently high, s_f for the proxy ray generation.

The actual proxy rays are then computed based on the resulting point distribution, P. To obtain proxy rays from P we consider all pairwise combinations (p', p''), with $p', p'' \in P$. During rendering a ray intersecting p' and p'' would be a line in world space and a curve in ξ space. Figure 3 shows an example of 9 rays going from one point of a face to all points of the opposite face. When connecting p' and p'' with a straight line, representing a viewing ray in world space, the proxy ray in ξ space can be obtained by sampling along the straight line and transforming each sample's position into ξ space. While this sampling is performed with a fixed but sufficiently high sampling rate s_r, it should be pointed out that it does not represent or determine the sampling rate used later during rendering. s_r is only used to obtain the geometry of the proxy rays, whereas the sampling rate used for the compositing during ray-traversal in the rendering step can still be chosen independently.

As the set of proxy rays, R, has to be dense its size is obviously a limiting factor. Assuming that our subdivision parameter, s_f, would result in n points along the u and the v coordinate of an element's face, we would have n^2 grid points on each face. Thus we would have $n^2 \cdot |F|$ grid points in total, assuming that each element has $|F|$ faces. This would result in $r_t = (n^2|F|)^2$ proxy rays to be precomputed. Though we have kept s_f constant to allow a less complex implementation of the ray-casting process, we have other possibilities to reduce the size of R. To achieve this reduction we exploit properties of the standard volume rendering integral which is used in the rendering stage:

$$L(x, \omega_o) = L_0 \cdot e^{-\int_{x_0}^{x} \kappa(x'')dx''} + \int_{x_0}^{x} c(x') \cdot e^{-\int_{x'}^{x} \kappa(x'')dx''} dx'$$

where $c(x')$ represents the emissive color at sample x' and $e^{-\int_{x_0}^{x} \kappa(x'')dx''}$ represents the exponential falloff between x_0 and x. When exploiting this rendering integral the same observations hold as when applying the integral within the context of regular volume rendering. For a ray cast through a medium the application of this integral results in an exponential falloff, i. e., when the ray is cast from one start point, x_0, to one end point, x_n, the samples in the regions surrounding x_0 will have considerably more influence on the image than those around x_n. In extreme cases where structures with a high degree of opacity are rendered, samples close to x_n do not have any influence at all. To select an optimal proxy ray subset during the precomputation stage we can take this observation into account. Therefore, we can use two different point distribution densities for entry and exit points, which is achieved by replacing s_f with s'_f and s''_f. Thus we obtain a point distribution of higher density, representing ray entry points, and a point distribution of lower density, representing exit points. When considering all entry and exit points of these two distributions in a pairwise manner, we achieve an importance-based ray sampling, where rays are more accurately represented when being close to the entry points. In comparison with the r_t rays computed with the uniform proxy ray distribution, the importance-driven ray reconstruction approach results in $r_i = (n_f \cdot |F|) \cdot (n_b \cdot |F|)$ rays, where n_f and n_b represent the number of precomputed entry and exit points. This results in a decreased number of proxy rays, where the proxy ray set R is more dense towards the respective entry points and thus better complies with the exponential falloff which occurs during ray-casting. This works best for rather homogeneous parameters, while in cases where all features of interest are located close to the faces a uniform proxy ray distribution is beneficial.

As s_f should be high but independent of the sampling rate used during rendering, it is unfeasible to store the proxy rays based on the precomputed samples. Under the assumption that the proxy rays are well-behaved and at least C^1 continuous, we can reduce the memory footprint by storing the proxy rays as parametric curves facilitating a lower number of control points than the original sampling density, s_f. We will present the ramifications of different numbers of control points for the error in Section 7. Due to the requirements regarding interpolation of the control points, local control, numerical stability, and non-complex evaluation, Catmull-Rom splines [5] are a natural choice for this parametric representation. Furthermore Catmull-Rom splines have built-in C^0 and C^1 continuity, thus being continuous in position as well as tangent vector, leading to a smooth curve. Additionally, Catmull-Rom splines allow us to obtain and interpolate the start and end tangent of the curved rays directly.

4.2 Curve Clustering

The curve compression, which is used to obtain a manageable set of proxy rays during rendering, first increases the similarity of the proxy rays by exploiting orientation and scaling invariance, before computing a representative subset using a clustering algorithm, whereby the increased similarity of the proxy rays allows us to compute a smaller subset of proxy rays while maintaining the same quality of compression.

Curve similarity. In the previous subsection, we have described how to obtain a densely populated set of proxy rays that are represented as parametric curves in ξ space. As this set is too large to be considered during rendering we will eliminate proxy rays from within this set by taking into account the following three observations which increase the similarity. First, a curve does not have a pre-defined entry and exit point order. As the ray traversal and the compositing are performed in the rendering stage we do not need to distinguish between two proxy rays with swapped entry and exit coordinates, if both rays are the same. The second helpful observation refers to the orientation of proxy rays in ξ space. As a proxy ray represents only the traversal path but not the dependent variables accessed during a traversal, we do not need to distinguish between two proxy rays which have the same overall geometry but are differently oriented in ξ space. As long as we know the entry and the exit point of a proxy ray we can orient the proxy ray's geometry such that it is correctly aligned in ξ space. This property is also related to the third observation, the length invariance, which we also exploit to increase the similarity during curve compression. As soon as we know the coordinates of the entry and the exit points we can not only adapt the orientation of a proxy ray but also its scaling. Therefore we do not have to distinguish between proxy rays which have the same geometry but differ only in the scaling factor along their principle axis.

We exploit the three observations stated above in order to achieve a higher degree of similarity during the subsequent clustering-based curve compression. To do so we transform the proxy rays into a common coordinate frame without compromising the uniqueness of each proxy ray as they comply to orientation and scaling invariance. The transformation is done in such a way that the entry point and the exit point lie on the z-axis, with the proxy ray's principal axis aligning with the same axis (see Figure 4). To be further able to define the rotation around this axis we take into account the first control point, c_i, that is not colinear with the start and end points and transform the proxy ray such that c_i lies in the yz-plane. If there is no such point, the proxy ray is a straight line so that it can be rotated to coincide with the z axis. After this transformation, M_{trans}, has been applied all proxy rays lie in the same coordinate system as shown in Figure 4, and thus the comparison performed during clustering produces a higher degree of similarity. While this transformation into a common coordinate system indirectly reduces the number of rays, it requires us to store the angle of rotation, θ, for each proxy ray. This is necessary as θ is required to be able to compute ${M_{trans}}^{-1}$, which is needed during rendering to correctly orient the precomputed proxy rays.

Curve clustering. To reduce the size of R we exploit the inter- and intra-element coherency discussed above. This allows us to perform clustering in order to obtain a representative subset of R, called R'. Several algorithms have been proposed for clustering curves and we have decided to exploit the approach proposed by Abraham et al. [1]. As this approach applies the clustering to the curve coefficients instead of the control points, it can be proven that the derived cluster representatives are optimal candidates. As we have previously increased the curve similarity of the set of proxy rays, R, by transforming them into a common coordinate system, we can use a lower number of cluster representatives when using the clustering method. This reduces the data size as well as computation time needed for the clustering. We will show in Section 7 how the number of clusters affects the achieved quality of the clustering.

Abraham et al. state that the comparison metric used during the clustering is the most crucial part, as it directly defines the similarity of two curves [1]. Inspired by their work we use the spanned area between two proxy rays as a metric. To compute the spanned area, we equidistantly sample the proxy rays with a high sampling rate, and generate a triangle strip between the sampling points. The similarity can then be expressed by a Riemann sum of the triangle areas. If we have two proxy rays, a and b, with their sampled points, $a_1,\dots,a_n$ and $b_1,\dots,b_n$, where due to the transformation M_{trans}, $a_1 = b_1$ and $a_n = b_n$ holds true, we can calculate the approximate area between the rays. If x_iy_j denotes the vector from x_i to y_j, then the similarity of a and b is proportional to:

$$\begin{aligned} 2\cdot d(x,y) \quad = \quad & \|a_1a_2 \times a_2b_2\| + \\ & \sum_{i=2}^{n-2} \|a_ia_{i+1} \times a_ib_i\| + \|b_ib_{i+1} \times b_{i+1}a_{i+1}\| + \\ & \|a_{n-1}a_n \times a_{n-1}b_{n-1}\|, \end{aligned}$$

It is clear that this function fulfills the metric traits of non-negativity, the identity of indiscernibles and symmetry, since all these properties hold for the area computation as well. Although we could have used the closed solution for the two curves and compute the analytically correct solution, we wanted to use the same metric for non-parametric curves as well. The same metric presented here will be used in the error analysis, where no closed solution for the bent ray is available.

For the actual clustering we exploit the k-means clustering algorithm which was originally presented by Hartigan [13]. It has the benefit that it is robust even with large high-dimensional data, and only needs the number of final clusters and a metric as parameters. Unfortunately the number of clusters is not known a priori and, in fact, there might be no definite answer as to which is the best number in general. To deal with this problem we have evaluated the impact of different numbers of clusters in Section 7. Alternatively, one could apply algorithms for finding an optimal number of clusters, such as for instance k-fold cross-validation, which is often used together with k-means. As output, k-means provides the desired number of representative proxy rays, one for each of the clusters. Besides the obtained cluster representatives we need to store for each proxy ray the cluster ID as well as the angle θ The cluster ID is used to determine the cluster to which it belongs.

5 Bent Ray-Casting

While Section 4 gives a comprehensive overview of the preprocessing steps needed to obtain the proxy ray set, R, and the clustered set, R', the actual rendering process is discussed in detail within this section. We perform GPU-based ray-casting by exploiting the data generated during the preprocessing stage, i.e., all cluster representatives as well as a list, for each proxy ray, storing the associated cluster ID and the rotation angle θ. As the elements are treated individually we need a way to be able to associate ray segments with the elements. Thus the rendering stage is divided into three substages. First element peeling which allows us to obtain the entry and exit points for each element. Second ray marching, where a ray in ξ space is reconstructed from the preprocessed data before it is traversed. Third ray interpolation, which is used to be able to deal with less dense proxy ray sets R. These three substages are discussed in the following three subsections.

5.1 Element Peeling

As the FE models consist of several elements they will, in general, overlap in image space. Therefore mechanisms are needed to determine which sequence of proxy rays best represents a specific pixel. To obtain this sequence of proxy rays we employ a modified depth peeling approach [19]. With this approach we can peel away, layer-by-layer, from the rendered FE model by using an additional depth comparison. When using depth peeling however, it is crucial to be able to distinguish between different fragments solely based on their depth values. Unfortunately, in our case this is not possible. As a FE model consists of several adjacent and touching elements, coinciding depth values are common and need to be handled robustly. As illustrated in Figure 5, two cases of depth coincidence occur. Point coincidences occur at the corners of adjacent elements (*top*) and surface coincidences occur where two adjacent elements' surfaces are touching (*bottom*). To deal with this issue we have implemented multi-layer depth peeling, where we take advantage of several attributes to resolve the depth coincidences without introducing errors. Figure 6 shows the employed information next to the color coded ξ and world space coordinates of the first two layers of a FE model. Besides the depth values we also take into account the current element's ID, as well as the ID of the current face in our peeling approach. Thus we can resolve the described coincidence cases and retrieve the entry and exit points for the current view ray segment. This allows us to fetch the cluster representative for this pair (see Section 6) and perform the ray marching.

5.2 Ray Marching

Once we know which proxy ray information needs to be accessed, we can reconstruct its ray geometry by retrieving the spline coefficients for the obtained cluster ID. The reconstructed spline is then

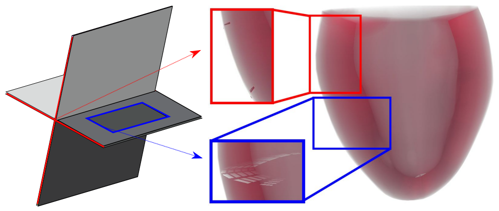
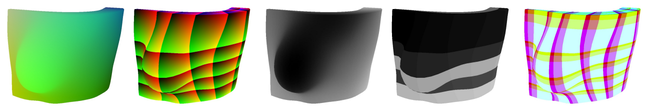


Fig. 5. Two cases of depth coincidence can occur, as illustrated (*left*), when rendering adjacent polygons. Point coincidence occurs at the corners of adjacent elements (*top*), while surface coincidence occurs at the touching elements' surfaces (*bottom*). All coincidences can be avoided (*right*).


(a) entry world (b) entry ξ (c) entry depth (d) entry element IDs (e) entry face IDs

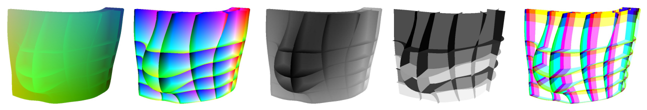
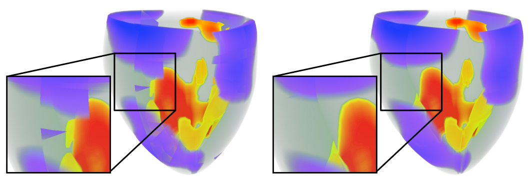

(f) exit world (g) exit ξ (h) exit depth (i) exit element IDs (j) exit face IDs

Fig. 6. Depth peeling is used to extract the layers of the FE model during rendering. The images show the world coordinates, the ξ coordinate, its depth, the element ID, and the face ID (*from left to right*). The first row shows the first depth layer and the second row shows the second depth layer.


(a) Precomputed proxy rays with $s_f = 2$

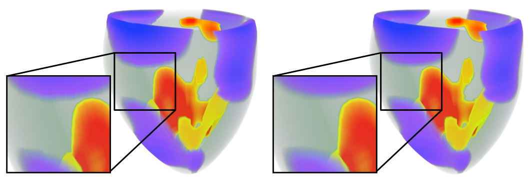

(b) Precomputed proxy rays with $s_f = 10$

Fig. 7. Results of our approach with two different parameter sets. The first row shows a precomputation with a grid parameter of $s_f = 2$ and the second with $s_f = 10$. Within each row we show the application of standard ray interpolation (*left*) as well as intra- and inter-ray interpolation (*right*). As shown in the close-ups, intra- and inter-ray interpolation reduces noticeable artifacts. Artifacts are still noticeable with 2×2 proxy rays, as shown in (a, *right*)

transformed back from the canonical orientation along the z-axis by applying $M_{trans}{}^{-1}$, such that the spline lies between the entry- and the exit-point of the actual viewing ray in world coordinates. Although the reconstructed spline is similar for pixels in a local neighborhood, the specific path is different because it is fitted to the specific entry and exit-point combination of the viewing ray. Once the proxy ray has been transformed back into its original location we can march along it and perform the actual bent ray-casting by fetching the dependent variables from the ξ space using our spline representation. The expensive world-to-material transformation that would be necessary in the straightforward approach is replaced by an evaluation of the precomputed spline representation.

To ensure that we consider possible high-frequencies in the material data, we need to ensure an equidistant sampling along the reconstructed ray in material space in order to obtain optimal results. The proxy rays, however, are parameterized by the parameter value $t \in [0,1]$ in world coordinates which does not take the shape of the ray in ξ space into account, i.e., given a spline s, $\|s(x) - s(y)\| \neq \|x - y\|$. That means that an equidistant sampling in world space will result in sampling points that are not equidistant along the curve in ξ space. As illustrated in Figure 8(a), we address this problem by using an arc length parametrization as presented by Guenter and Parent [12]. This leads to a function $t \mapsto t'$. If the sampling is done using t' instead of t, the sampled values will be equidistant along the curve in ξ space. To ensure accuracy w.r.t. the rendering integral, we employ the equidistant spacing in ξ space to access the dependent variables, but use the distance in world space for the opacity computation.

The subdivision of a curve ray penetrating the whole FE model into segments, based on the penetrated FEs, requires an appropriate border handling scheme. If we sample the ray equidistantly, in a naïve way, the last point on the ray, which lies on the exit face of a FE, will, in general, not coincide with an intended sampling point. Special consideration is needed when handling a ray, a, that is divided into two segments, r and s, in such a way that $r_n = s_0$, i. e., the last sample point of r is the first sample point of s. In this situation we would have an equidistant sampling rate of $|a_i - a_{i+1}| = \delta$ for the two rays. However, the naïve implementation will result in $|r_{n-1} - r_n| \neq \delta$. To remedy this, we do not sample the last point on each ray but save the distance $|r_{n-1} - r_n|$ instead and use it as an offset for the first sampling point in the next ray segment (see Figure 8(b)).

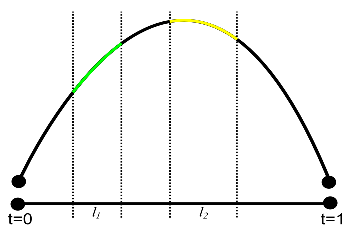
5.3 Ray Interpolation

In order to improve the image quality of our ray-casting approach, we introduce two specialized interpolation schemes: *inter-ray interpolation* and *intra-ray interpolation*. With the inter-ray interpolation we use the four nearest proxy rays instead of the "nearest-neighbor" proxy ray for a given pixel position, and interpolate between them. This interpolation is carried out by using bilinear interpolation at every sample position along the viewing ray. The interpolation factors for these bilinear interpolations are based on the factors derived for the entry and the exit face. The intra-ray interpolation, instead, employs swapping of entry and exit points in order to obtain a mirrored copy of the current ray. Thus we can interpolate between the original ray at position t and the mirrored ray at position $t' = 1 - t$ while traversing the initially obtained ray. Both of these interpolation schemes improve the image quality in situations where only a low resolution is available, as we show in Figure 7.

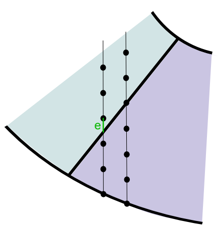
6 Implementation

In this section, we explain all relevant implementational details regarding concepts mentioned previously.

Proxy Ray Generation. In order to have a consistent memory layout we will use the same data structure to store the computed control points here as we will use later in the rendering stage. There are two demands on the data structure: given the parameters for the stored value, access should be in $O(1)$ and it should have a rectangular structure to be efficiently usable at GPU level. The parameters necessary for describing each proxy ray uniquely are (e, f_s, f_e, p_s, p_e), with e being the element number, p_s the position on the entry face, f_s, and p_e the position on the exit face, f_e. Let $|E|$ denote the total number of elements and $|F|$ be the number of faces per element. Furthermore, recall that s'_f and s''_f, as defined in Section 4.1, are the subdivision parameters that control the number of points on the entry and exit face



(a) Equidistant sampling



(b) Boundary handling

Fig. 8. (a) To achieve equidistant sampling along a ray in ξ space (*top*), an arc length parametrization is applied resulting in a non-uniform sampling in world space (*bottom*). (b) Rays penetrating multiple elements must be processed such that constant sampling step sizes are obtained at the borders. The overshoot, e, of the first element can be used to offset the first sampling point in the next element.

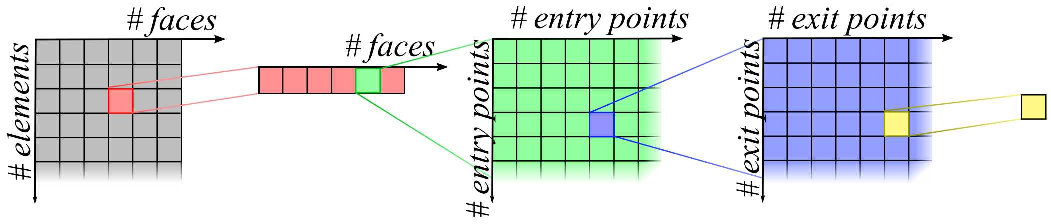


Fig. 9. Memory layout used to gain access to the precomputed proxy ray information. To retrieve the information required during rendering in constant time, a hierarchical subdivision is used. For each proxy ray, we store the cluster ID as well as the rotation angle θ.

respectively. It then holds true that $p'_f \in [0, s'_f]$ and $p_e \in [0, s''_f - 1]$. Figure 9 shows the memory layout that we have chosen for this setting. At the highest level of the hierarchy we arrange the data based on f_s along the x-axis and e along the y-axis. Thus we obtain $|F| \cdot |E|$ cells which contain the precomputed information for rays entering e through f_s. Each of these cells is partitioned based on the f_e along the x-axis, such that the resulting cells contain only the information for those rays exiting through f_e. We again partition the cells based on the number, s'_f, of precomputed points on the entry face. Finally the obtained cells are partitioned based on the number, s''_f, of precomputed points on the exit face. The memory layout depicted here is represented by the two following equations which allows us to store all data in a rectangular manner:

$$p_x = (f_s \cdot (s'_f \cdot s''_f \cdot |E|)) + (f_e \cdot (s'_f \cdot s''_f)) + (p_{s,x} \cdot s''_f) + p_{e,x} \quad (1)$$

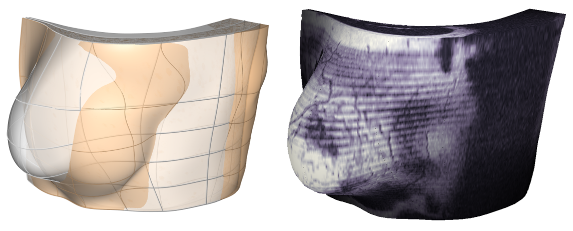
$$p_y = (e \cdot (s'_f \cdot s''_f)) + (p_{s,y} \cdot s''_f) + p_{e,y} \quad (2)$$

Curve Similarity. The first step here is to increase the similarity of the stored proxy rays in such a way, that the subset, R', is as small as possible. Three transformations are performed after which each proxy ray fulfills three traits. 1. The starting point, x_0, is at $(0,0,0)$. 2. The first non-colinear point x_i lies in the yz-plane. 3. The end point, x_n, is at $(0,0,1)$. We apply three matrices M_t, M_{rot}, and M_s to all control points, where M_t is a translation by $-x_0$, M_s is a scaling by $1/(x_n.z)$, and M_{rot} is a rigid-body transformation that will rotate the points such that x_n is on the z axis and the non-colinearity is fulfilled. The multiplication $M_s \cdot M_{rot} \cdot M_t$ is equal to M_{trans} as mentioned previously. Thus we get new control points $x' = M_s \cdot M_{rot} \cdot M_t \cdot x$ with a higher degree of similarity between proxy rays. By enforcing that x_i lies in the yz-plane we lose the rotational angle w.r.t. the z-axis. We will store this angle θ for each proxy ray separately so that we can access it together with the cluster id.

Curve clustering. In order to reduce the number of proxy rays which need to be stored on the graphics card, we apply a clustering algorithm such that multiple rays are represented by a single representative. Finding a good number of iterations for the applied K-means algorithm a priori is a hard problem, so we investigated multiple strategies. The simplest, and already feasible, strategy to use is a fixed number of iterations (see Figure 13(b)). Another possible solution is to trace the percentage of splines that change clusters in a single iteration, based on the assumption that this number decreases asymptotically to 0. A CPU implementation using C++ and OpenMP was used to generate the measurements presented in Section 7. The clustering was performed on 8 threads and it was found that the hard disk access, as well as the computing power, are bottle necks for faster execution times. Additionally, an OpenCL-based implementation was tested that keeps the cluster representatives for the current iteration in local memory and fetches the splines from global memory.

Ray Marching. The rendering step starts with a depth peeling step to get the entry- and exit points for each view ray. In general we set the additional depth test to accept all fragments which have a greater depth than or equal to that of the current fragment. To resolve the two coincidence cases, as presented in Section 5.1, we ensure that each pair of entry- and exit-points has the same element ID. This, together with ensuring different face IDs, is enough to correctly resolve the point and the surface coincidence (see Figure 5). This additional information also leads to a more stable algorithm w.r.t. the available accuracy of the depth buffer. We tested depth buffer accuracies of 16 and 24 bits but found no noticeable visual differences. The approach also works for several situations where the entry and exit points lie on the same surface, as these points in general have unequal depth values thereby making the test for face inequality unnecessary. In the case of a silhouette edge, where two fragments coincide in the same depth value and originate from the same element, we do not perform a depth peeling but transform only the entry point and obtain a single sample.

The geometry representation of the FE models contains 4 values for each node; the three color channels provide the ξ value (see Figures 6(b) and (g)) and the alpha channel stores the element for which the nodal value is defined (see Figures 6(d) and (i)). Based on this information, all free parameters for Equations 1 and 2 can be derived. Thus, the cluster can be fetched for this entry-exit point pair and the ray marching can be performed on the spline.



(a) surface rendering (b) volumetric rendering

Fig. 10. FE breast model with MRI material space rendered as surface representation showing the deformed and the undeformed surface (a), and showing the deformed volume with our interactive volumetric approach (b).

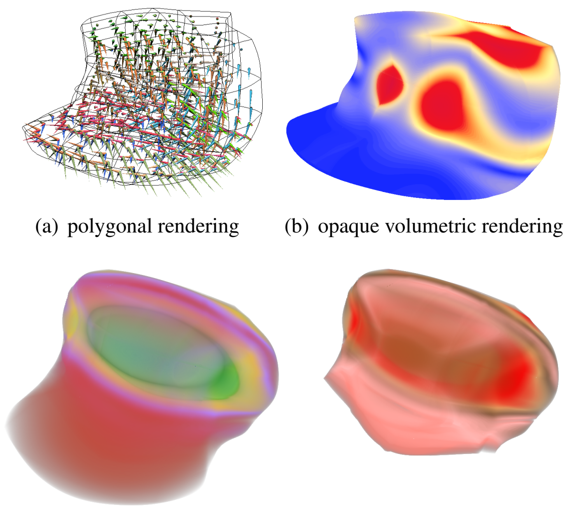
Element Generalization. The examples in this paper all use elements with cuboidal parent elements. In order to apply the technique to differently shaped elements, e.g., tetrahedral elements, methods must be implemented for depth peeling and sampling the element faces. This is straight forward. In fact, the current implementation already works for differently shaped elements, e.g. where multiple vertices of the parent element are mapped to the same vertex in world coordinates. This is the case for the apex of the left ventricular model, where the resulting elements have a curved wedge-like shape.

7 Results

To assess the quality and the performance of the proposed visualization technique we have applied it to different data sets. Below we discuss the visual results, the introduced errors and, the performance of our approach as well as the memory requirements.

7.1 Visual Results

To demonstrate the visual outcome of the proposed concepts, we have applied them to different FE models which all consist of elements with curvilinear boundaries. These vary with respect to the number of cells and the order of the used interpolation functions. Although we include only interpolation functions of a degree-7 or 9 polynomial, our method is not limited to any specific order as long as the coherencies are fulfilled. In the following paragraphs we describe these models and the



(a) polygonal rendering (b) opaque volumetric rendering

(c) translucent volumetric rendering

Fig. 11. FE tongue model rendered as a polygonal representation, (a), and with our interactive volumetric approach, (b) and (c).

visual results achieved.

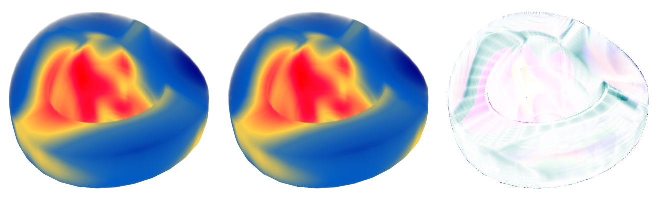
Breast model. We have applied our approach to a deformation model of a breast data set, consisting of 60 tricubic elements, acquired from an MRI scan. The dependent variables in this case are the actual intensities of the MRI field, and the FE modeling has been applied to predict the supine shape of the data set originally acquired in the prone position. Figure 10 shows a comparison of the supine position rendered with our interactive volumetric approach, (b), as compared with a standard surface based representation, (a).

Tongue model. Furthermore we have applied our approach to a model of a human tongue, consisting of 64 tricubic elements, with simulated tissue densities and muscle fiber directions. In this case each node stores one coordinate vector, and seven derivatives and mixed derivatives for the tensor product of the cubic Hermite interpolation functions in all three coordinate directions. The results, as compared with a polygonal representation, are shown in Figure 11. In this case in particular, the multi-parametric capabilities of our algorithm are beneficial as the tongue model contains a relatively high number of parameters. When one would apply resampling to all of these parameter volumes instead, this would result in an unmanageable amount of data for a reasonable resolution.

Heart model. A heart data set, which consists of 16 bicubic-linear elements (i.e. two interpolation functions are of cubic order, the other one is linear), where the strain values for each point have been computed from displacements obtained from tagged MRI data, serves as the third example (see Figure 1). For our visualization we have taken into account the three principal strain directions, which are the eigenvalues of the tensor. The principle strains include the maximum and minimum deformation and the corresponding eigenvectors give the direction of these deformations. These directions are circumferential, longitudinal and radial.

7.2 Error Analysis

Our rendering approach contains several steps which potentially introduce visual errors. First, the proxy ray generation explained in Subsection 4.1 introduces an error based on the number of actual proxy rays. The visual impact of this error is analyzed in Figure 7. In the two rows, different numbers of proxy rays have been computed. In the top row a very low number, with 2×2 proxy rays, per element face has been chosen whereas the bottom row uses 10×10 proxy rays. As can be seen, the error introduced by a low number of proxy rays is clearly visible. However, when using 10×10 proxy rays no artifacts are visible when using the proposed ray interpolation mechanism. As the clustering itself also introduces an error, we have analyzed the impact in



(a) straightforward approach (b) our approach (c) visual difference (emphasized 10 times)

Fig. 12. Visual error as obtained when applying our technique. A straightforward volumetric approach (a) and our interactive volumetric approach (b) have been compared, which results in the error image exaggerated tenfold (c).

Figure 13(b), which shows the error for a varying number of clusters. It is obvious that the error becomes smaller if we use more clusters. The number of clusters can, depending on data set, be quite low in relation to the number of proxy rays as seen in Figure 13(c). While these two analyses show the error for the proxy ray computation and the clustering separately, we have also compared the results achieved by our algorithm with the ground truth, a straightforward raycaster as described in Section 4, which evaluates the world-to-ξ transformation during each sample step. We chose this method as a comparison, because it produces the smallest error by itself and is thereby suited to demonstrate the error introduced by our method. As can be seen in Figure 12, the visual error is rather low, as only a tenfold exaggeration of the generated error image makes it noticeable.

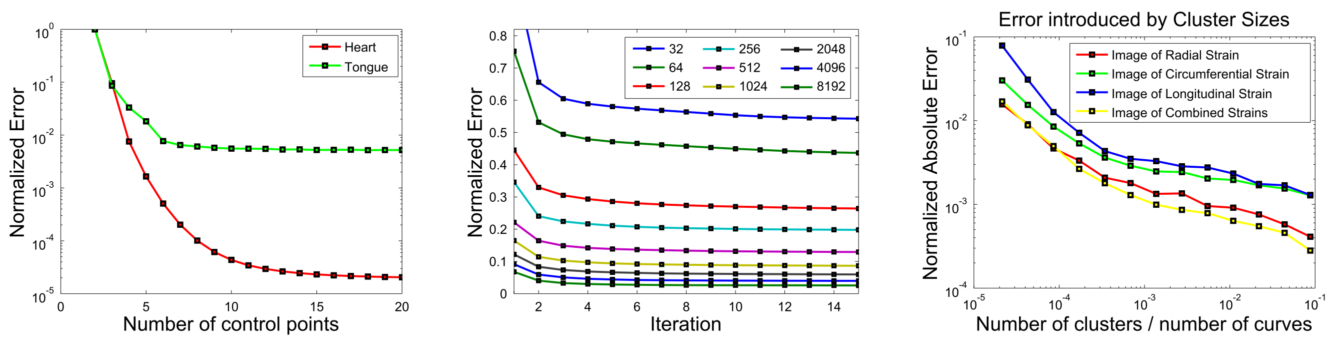
We described in Section 4.1 that we can use a varying number of control points to represent the proxy ray. This, of course, is an approximation of the original bent ray we get by using the straightforward implementation. Therefore we analyze the effect of this parameter has the error introduced by the representation. Figure 13(a) shows this error, computed by comparing the spline representation to the ray where each sample point is transformed. We sampled every ray at a sufficiently high rate of 1024 points and computed the area metric as described in Section 4.2. The abscissa shows the number of control points representing the proxy ray, where 2 points denote a straight line. For easier comparison, all error values were normalized with respect to this value. As the error is dependent on the shape of the FE, we computed the error for all proxy rays in the *Heart* and *Tongue* models.

7.3 Performance Analysis

We have to differentiate between the precomputation and the rendering step in the performance analysis. Although the time used for precomputation is dependent on both the proxy ray computation and the

Table 1. Performance analysis (in fps) of our technique for different grid sizes compared with a straightforward (SF) implementation. The setups match the first three subfigures in Figure 1: images showing radial strain, circumferential strain, and longitudinal strain.

Figure 1	Pixel	GPU	Our Technique [s_f, Clusters]		
TF Setups	Res	(SF)	$3, 1024$	$5, 2048$	$10, 4096$
Radial	256^2	2.58	14.52	14.25	14.05
Strain	512^2	1.33	11.40	11.71	11.02
	1024^2	0.41	7.09	7.02	6.83
Circum-	256^2	2.60	14.63	14.81	14.21
ferential	512^2	1.30	12.09	11.85	11.72
Strain	1024^2	0.40	7.15	7.12	7.10
Longi-	256^2	2.53	14.66	14.70	14.74
tudinal	512^2	1.35	11.78	11.78	11.83
Strain	1024^2	0.40	7.12	7.12	6.88



(a) Error introduced by approximating the bent ray by a spline

(b) Error introduced by clustering

(c) Image error measured for varying cluster sizes

Fig. 13. **Error measurements.** (a) shows the error introduced by the approximation of the correct bent ray by sampling a spline. The abscissa shows the number of control points used to construct the spline, where 2 points represent a straight line. (b) shows the error for varying numbers of clusters depending on the number of iterations after which the k-means algorithm has stopped. Based on the heart data set with a grid resolution of 5×5, which resulted in 360.000 rays. (c) shows the normalized absolute pixel error (background pixels are neglected) plotted against the percentage of clusters relative to the total amount of curves. The calculations have been performed on the subfigures of Figure 1, thus varying the transfer function and strain direction. All error values were normalized w.r.t. the respective highest error.

clustering, the latter takes more time than the former. In our unoptimized CPU implementation, the clustering for a 5×5 proxy ray setup with 8192 clusters takes roughly 4-6 hours, while a 15×15 setup with 8192 clusters takes approximately 72-84 hours of time. We managed to reduce this time with an unoptimized OpenCL implementation by an order of magnitude to about 20-25 minutes for the 5×5 proxy ray setup with 8192 clusters, which is in line with former works [22]. The clustering and all performance tests were executed on an Intel Xeon Quad-Core 3.06 GHz with 6 GB of working memory, equipped with a Geforce 580 card with 1 GB of memory.

Table 1 shows the rendering performance of the presented volumetric FE visualization technique, as compared with the straightforward volumetric GPU implementation. As can be seen, the performance of our technique is affected by image resolution and independent of the number of clusters and proxy rays. The rendering quality is thus affected by a trade-off between the number of proxy rays used during the precomputation and the number of clusters used during the curve compression. In the worst case tested our technique provides above 7 fps instead of 0.4 fps. While the 14.81 fps achieved with our method in the best tested case enabled full interactivity, the 7 fps also resulted in an interactive experience.

7.4 Memory Requirements

A limiting factor of the presented method is the memory that is required to store the proxy rays, angles and cluster IDs. Besides the computational time, the available graphics memory is the only limitation to the number of elements, grid resolution, and number of clusters. In this section, we will consider the impact of each of these factors on the memory requirements using the same definitions as in Section 6 with the addition of c as the number of clusters. The total number of rays per element is given by $r_t = (s'_f \cdot s''_f \cdot |F|)^2$ which leads to $e \cdot r_t \cdot s_r$ points that have to be transformed and stored. It should be noted that this data does not need to fit on the graphics card and is only an intermediate result. The required memory after the clustering has been performed can be estimated by: $c \cdot s_r + e \cdot r_t \cdot 2$. The first part of the equation is the storage necessary for the clustered control points of the representatives. The second part is the lookup table as described in Section 5.1 and needs to store both the cluster ID as well as the angle. A significant restriction on our method is obvious in the latter formula. The second term grows linearly in the number of elements and poses, therefore, a limitation on the mesh sizes we are able to render. Example values for the heart are given in Table 2, when we assume float precision, i.e., every point will occupy 12 bytes of memory, and 5 stored points per cluster.

Table 2. Memory consumption of data structures during rendering for varying grid resolutions and number of clusters on the heart dataset.

s_f	Without Clustering	With Certain Number of Clusters 512	2048	4096
3	2.7 MB	0.39 MB	0.47 MB	0.82 MB
8	135 MB	18 MB	18.1 MB	18.5 MB
10	329 MB	43.9 MB	44.0 MB	44.1 MB
15	1.62 GB	222.5 MB	222.6 MB	222.9 MB

8 Conclusions & Future Work

In this paper we have presented a ray-casting based approach that enables the interactive exploration of high-order FE models. By shifting the computationally complex world-to-material transformation from the rendering stage into a preprocessing stage we are able to achieve interactive frame rates and thus allow for an interactive, in-detail, multi-parametric inspection of FE models using transfer functions and other parameters on standard GPUs. We have demonstrated the outcome of the presented approach by applying it to several real-world data sets and analyzing the errors introduced by our approach. Furthermore, we have discussed the rendering performance gain as well as the memory requirements.

There are several opportunities to further improve the presented concepts in the future. When precomputing the proxy rays we currently use a equidistant layout for the generation of the set, P, of uniformly distributed start and end points. In comparison with such an equidistant layout more elaborate sampling schemes, as used in many areas of computer graphics, might lead to an improved quality with the same number of proxy rays. While our approach currently supports scalar data only, in the future we plan to use the proposed method to display vector and tensor data defined over ξ space.

Acknowledgments

We thank all reviewers for their fruitful comments which helped to greatly improve this paper. We would also like to thank Martyn Nash, Poul Nielsen, and Alistair Young, for providing the data sets we used in this work, and Matt Cooper for proof reading. This work was partly supported by grants from the Swedish Research Council (VR, grant 2011-4113), the Excellence Center at Linköping and Lund in Information Technology (ELLIIT) and the Swedish e-Science Research Centre (SeRC). The presented concepts have been realized using the Voreen open source visualization framework (www.voreen.org).

References

[1] C. Abraham, P. Cornillion, E. Matzner-Løer, and N. Molinari. Unsupervised curve clustering using b-splines. *Scandinavian Journal of Statistics*, 30, 2003.

[2] J. Berkley, G. Turkiyyah, D. Berg, M. Ganter, and S. Weghorst. Real-time finite element modeling for surgery simulation: An application to virtual suturing. *IEEE Transactions on Visualization and Computer Graphics*, 10:314–325, 2004.

[3] C. Bradley, A. Pullan, and P. Hunter. Geometric modeling of the human torso using cubic hermite elements. *Annals of Biomedical Engineering*, 1997.

[4] M. Brasher and R. Haimes. Rendering planar cuts through quadratic and cubic finite elements. In *IEEE Visualization*, 2004.

[5] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, 1974.

[6] C. Everitt. Interactive order-independent transparency, 2001.

[7] R. Farias, J. S. B. Mitchell, and C. T. Silva. Zsweep: an efficient and exact projection algorithm for unstructured volume rendering. In *IEEE Volume Visualization*, pages 91–99, 2000.

[8] M. P. Garrity. Raytracing irregular volume data. *SIGGRAPH Comput. Graph.*, 24(5):35–40, Nov. 1990.

[9] L. Gelberg, D. Kamins, D. Parker, and J. Sacks. Visualization techniques for structured and unstructured scientific data. In *ACM SIGGRAPH 1990 Courses*, SIGGRAPH '90. ACM, 1990.

[10] J. Georgii and R. Westermann. A generic and scalable pipeline for gpu tetrahedral grid rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12:1345–1352, 2006.

[11] S. Grimm, M. Meissner, A. Kanitsar, and M. E. Gröller. Parallel peeling of curvilinear grids. Technical Report TR-186-2-04-07, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2004.

[12] B. Guenter and R. Parent. Computing the arc length of parametric curves. *IEEE Computer Graphics & Applications*, 1990.

[13] J. Hartigan. *Clustering Algorithms*. New York: Wiley, 1975.

[14] L. Hong and A. Kaufman. Accelerated ray-casting for curvilinear volumes. In *Proceedings of the conference on Visualization '98*, pages 247–253, 1998.

[15] L. Hong and A. E. Kaufman. Fast projection-based ray-casting algorithm for rendering curvilinear volumes. *IEEE Transactions on Visualization and Computer Graphics*, 5:322–332, October 1999.

[16] G. E. Karniadakis and S. J. Sherwin. *Spectral/hp Element Methods for CFD*. Oxford University Press, 1999.

[17] P. M. Knupp. Remarks on mesh quality. *American Institute of Aeronautics and Astronautics*, 2007.

[18] A. W. C. Lee, J. A. Schnabel, V. Rajagopal, P. M. F. Nielsen, and M. P. Nash. Breast image registration by combining finite elements and free-form deformations. In *Proceedings of the 10th International Workshop on Digital Mammography (IWDM 2010)*, pages 736–743, 2010.

[19] A. Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl.*, 9(4):43–55, July 1989.

[20] G. Marmitt, H. Friedrich, and P. Slusallek. Recent advancements in ray-tracing based volume rendering techniques. In *Proceedings of 10th International Fall Workshop - Vision, Modeling, and Visualization (VMV) 2005*, pages 131–138, 2005.

[21] S. Martin, H.-W. Shen, and R. Samtaney. Efficient rendering of extrudable curvilinear volumes. In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, 2008.

[22] C. Meß and T. Ropinski. Efficient Acquisition and Clustering of Local Histograms for Representing Voxel Neighborhoods. In *IEEE/EG Volume Graphics*, pages 117–124, 2010.

[23] M. Meyer, B. Nelson, R. Kirby, and R. Whitaker. Particle systems for efficient and accurate high-order finite element visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2007.

[24] K. Moreland and E. Angel. A fast high accuracy volume renderer for unstructured data. In *IEEE Symposium on Volume Visualization and Graphics*, 2004.

[25] B. Nelson, R. Haimes, and R. Kirby. Gpu-based interactive cut-surface extraction from high-order finite element fields. *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[26] B. Nelson and R. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2006.

[27] W. H. Press, W. T. Vetterling, S. A. Teukolsky, and B. P. Flannery. *Numerical Recipes in C - The Art of Scientific Computing*. Cambridge University Press, 1992.

[28] D. M. Reed, R. Yagel, A. Law, P.-W. Shin, and N. Shareef. Hardware assisted volume rendering of unstructured grids by incremental slicing. In *Proceedings of the 1996 symposium on Volume visualization*, VVS '96, pages 55–ff., Piscataway, NJ, USA, 1996. IEEE Press.

[29] D. Rose and T. Ertl. Interactive visualization of large finite element models. In *Proceedings of the Workshop on Vision, Modelling, and Visualization VMV '03*, pages 585–592, 2003.

[30] W. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. Pebay, R. O'Bara, and S. Tendulkar. Methods and framework for visualizing higher-order finite elements. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):446–460, 2006.

[31] C. Schwab. *p- and hp- Finite Element Methods*. Oxford University Press, 1999.

[32] N. Smith, C. Stevens, and P. Hunter. Computational modeling of ventricular mechanics and energetics. *Applied Mechanics Reviews*, 58:77–90, 2005.

[33] B. Szabó and I. Babuska. *Finite Element Analysis*. Wiley-Interscience, 1991.

[34] M. Üffinger, S. Frey, and T. Ertl. Interactive high-quality visualization of higher-order finite elements. *Computer Graphics Forum*, 29(2):337–346, 2010.

[35] J. Wihelms, J. Challinger, N. Alper, S. Ramamoorthy, and A. Vaziri. Direct volume rendering of curvilinear volumes. *SIGGRAPH*, 24:41–47, 1990.

[36] D. Wiley, H. Childs, B. Hamann, and K. Joy. Ray casting curved-quadratic elements. *IEEE Transactions on Visualization and Computer Graphics*, 2004.

[37] B. Wünsche. A toolkit for visualizing biomedical data sets. In *GRAPHITE '03*, pages 167–175, 2003.

[38] B. C. Wünsche and A. A. Young. The visualization and measurement of left ventricular deformation using finite element models. *JVLC - Biomedical Visualization for Bioinformatics*, 14(4):299–326, 2003.

[39] A. A. Young, B. R. Cowan, S. F. Thrupp, W. J. Hedley, and L. J. Dell'Italia. Left ventricular mass and volume: Fast calculation with guide-point modeling on MR images. *Radiology*, 216:597–602, Aug. 2000.