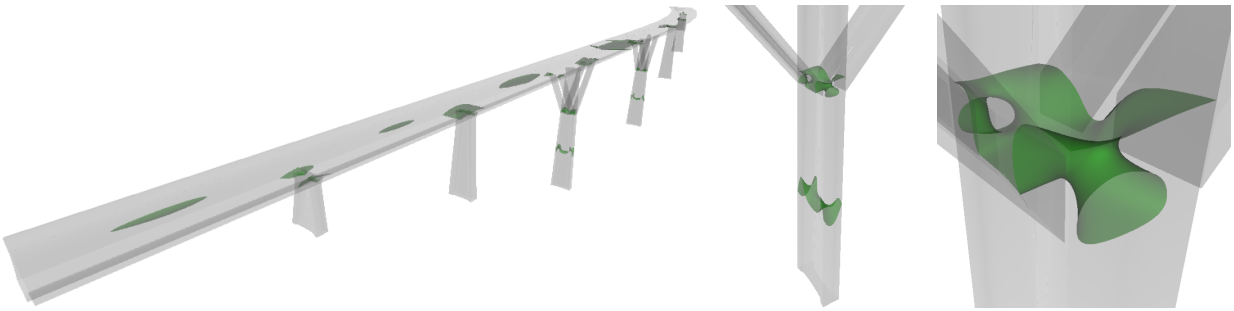
Direct Isosurface Ray Casting of NURBS-based Isogeometric Analysis

Andre Schollmeyer and Bernd Froehlich, *Member, IEEE*



(a) (b) (c)

Fig. 1. The images show the pixel-accurate isosurface visualization of the strain of this bridge model. For these views, the frame rates range from 18Hz for the total view (a) to 9Hz for the close-up on the pier (c). Users can interactively explore the model by adjusting the desired isovalue or navigating to different points of interest.

Abstract— In NURBS-based isogeometric analysis, the basis functions of a 3D model's geometric description also form the basis for the solution space of variational formulations of partial differential equations. In order to visualize the results of a NURBS-based isogeometric analysis, we developed a novel GPU-based multi-pass isosurface visualization technique which performs directly on an equivalent rational Bézier representation without the need for discretization or approximation. Our approach utilizes rasterization to generate a list of intervals along the ray that each potentially contain boundary or isosurface intersections. Depth-sorting this list for each ray allows us to proceed in front-to-back order and enables early ray termination. We detect multiple intersections of a ray with the higher-order surface of the model using a sampling-based root-isolation method. The model's surfaces and the isosurfaces always appear smooth, independent of the zoom level due to our pixel-precise processing scheme. Our adaptive sampling strategy minimizes costs for point evaluations and intersection computations. The implementation shows that the proposed approach interactively visualizes volume meshes containing hundreds of thousands of Bézier elements on current graphics hardware. A comparison to a GPU-based ray casting implementation using spatial data structures indicates that our approach generally performs significantly faster while being more accurate.

Index Terms—Computer graphics, Data Visualization, Isosurfaces, NURBS, Ray Casting.

1 Introduction

The concept of isogeometric analysis (IGA) is a promising attempt to close the gap between computer aided design (CAD) and finite element analysis (FEA). This gap exists because the model representation used for design is generally not suitable for finite element analysis and a polygonal or piecewise polynomial approximation of the actual geometry is used instead. The generation of such an approximation is time-consuming and the subsequent iteration between two different model representations is heavily involved and error prone.

As a solution, NURBS-based isogeometric analysis employs a trivariate NURBS (Non-uniform rational B-splines) representation which is based on the exact CAD geometry. The geometric flexibility and the inherent higher-order continuity of the NURBS basis are both significant advantages to standard finite element technology. In particular, they allow for an exact representation of a much larger class of objects, such as conic shapes, and prove beneficial for problems in which smoothness of the simulated properties is of great importance (e.g. in fluid mechanics). In the analysis, the solution space of the dependent variables is represented in terms of the same basis functions used for the geometric representation.

While modeling and simulation in isogeometric analysis are based on exactly the same geometric representations, there are no interactive visualization methods that perform directly on the trivariate NURBS volume mesh. To complete this isogeometric pipeline, we developed a GPU-based ray casting approach for the direct isosurface visualization of NURBS-based isogeometric analysis. In a preprocessing stage, we convert the NURBS representation into an equivalent rational Bézier representation in order to generate local bounds of the simulated variables. At runtime, these bounds are used to focus the search for isosurfaces on the relevant parts of the volume. Our multi-pass approach exploits current graphics hardware capabilities for efficient generation and sorting of Bézier cell intersection intervals along all rays which need further computation. This ray-interval generation scheme inherently allows for front-to-back processing of the volume elements and early ray termination. For each interval, ray entry and exit points for the associated Bézier cells are computed. Once these points are identified, we search the corresponding part in a cell for isosurface intersections by sampling along the ray. The computational costs of this expensive operation are significantly reduced through the use of an adaptive sampling strategy and an equally adaptive error metric.

Due to the historical dominance of finite element methods and the

• *Andre Schollmeyer and Bernd Froehlich are with the Virtual Reality Systems Group at the Bauhaus-Universität Weimar. E-mail: andre.schollmeyer@uni-weimar.de and bernd.froehlich@uni-weimar.de.*

Manuscript received 31 Mar. 2014; accepted 1 Aug. 2014; date of publication xx xxx 2014; date of current version xx xxx 2014.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

novelty of the isogeometric approach, there has been little attention paid to developing direct visualization algorithms for such higher-order NURBS-based volume representations. However, the recent advancements of direct rendering methods for bivariate NURBS surfaces, which also employ ray casting [19][23][28], motivated our approach. Our idea was to extend this practice to the task of interactive visualization of trivariate NURBS volumes and perform ray casting on the GPU using the parametric representation. Ray casting higher-order volume representations generally requires finding the intersection between the ray and the curved boundary of the volume. In contrast to other approaches, our system does not sidestep the issue of multiple ray-face intersections, but instead provides a practical and robust solution using a sampling-based root isolation approach.

We present a novel GPU-based algorithm for direct isosurface rendering of a NURBS-based isogeometric analysis. The central properties of our algorithm are: minimal preprocessing costs, compact storage and pixel-accurate visualization as a result of the direct use of the parametric description. All rays are processed in front-to-back order, which allows for early ray termination. Our main contributions are:

- A novel ray-generation scheme that creates only ray segments which potentially contribute to the final image
- A robust approach for finding all intersections between a ray and the curved boundary of the model making use of the smooth trivariate tensor-product nature of our models
- An effective solution for the memory-allocation bottleneck, which is a typical issue when constructing per-pixel lists on the GPU

Our algorithm outperforms current state-of-the-art isosurface rendering approaches for higher-order volume representations due to its output-sensitive processing scheme. Our highly optimized GPU-based implementation maintains interactive frame rates for models containing hundreds of thousands high-order volume elements. We compared our approach to conventional GPU-based ray casting implementations using spatial data structures. The results show that our algorithm typically performs significantly faster due to a lower number of generated rays, better cache coherence, avoiding segmentation issues and processing only the relevant cells of the model.

2 NURBS-Based Isogeometric Analysis

The methodology of isogeometric analysis as proposed by Hughes et al. [12] is a computational technique which generalizes standard finite element methods. The concept refrains from generating a separate finite element mesh for analysis purposes, but instead makes use of the exact NURBS geometry — the standard representation in most CAD systems. NURBS offer a compact representation and are well suited as an accurate description for smooth geometries. The inherent property of higher-order continuity is advantageous over C^0-continuous finite elements and is highly desired in the field of fluid mechanics [2] and structural analysis [10].

A NURBS-based isogeometric analysis operates on solid geometry. Thus, CAD modeling needs to deliver solids or the surface geometry has to be used to generate NURBS solids in a post-process. This process is obviously a non-trivial task and is not addressed in this paper.

The isogeometric analysis performs on an unstructured or partially structured mesh of NURBS solids. A single trivariate NURBS solid $\mathbf{V}$ of degree (l,m,n) is defined by

$$\mathbf{V}(u,v,w) = \frac{\sum_{i=0}^{o}\sum_{j=0}^{p}\sum_{k=0}^{q} N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)w_{i,j,k}\mathbf{p}_{i,j,k}}{\sum_{i=0}^{o}\sum_{j=0}^{p}\sum_{k=0}^{q} N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)w_{i,j,k}} \qquad (1)$$

where $N_{i,l}$, $N_{j,m}$ and $N_{k,n}$ are the B-Spline basis functions and $\mathbf{p}_{i,j,k}$ are points of a control net with the dimensions $(o+1)\times(p+1)\times(q+1)$ and the corresponding weights $w_{i,j,k}$. The B-Spline basis functions are given by a set of knot vectors, which also define the domain $\Omega \subset \mathbb{R}^3$ of the NURBS volume. The coordinates in the domain are denoted by $\mathbf{u} = (u,v,w)^T$, $\mathbf{u} \in \Omega$. For simplicity, all estimates given in this work use an identical polynomial degree n.

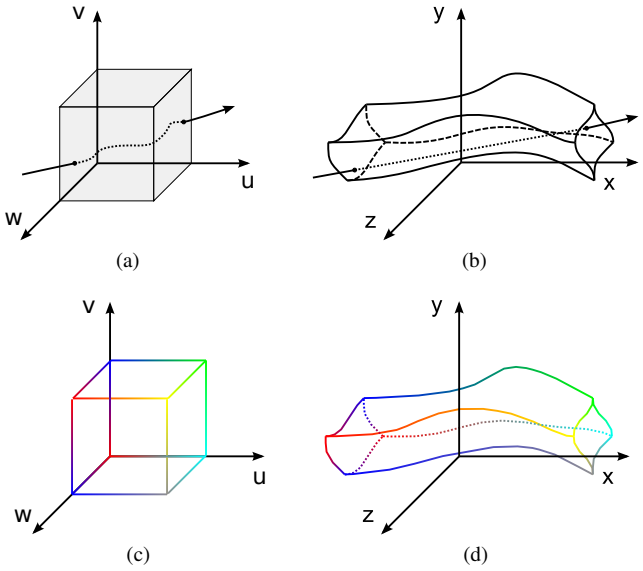


Fig. 2. This Figure shows the mapping (d) of attribute values (c), which are color coded, onto a single NURBS solid (b). Both descriptions are based on the same domain (a).

The simulation parameters of the analysis depend on the subject the method is applied to. The output of a fluid simulation may be pressure and flow, while structural computations typically involve stress, strain and displacement. In this work, we generalize and denote by an attribute any parameter that has been subject to the analysis.

In general, the solution of a simulation is a function $\Psi : \Omega \to \kappa$ that maps from the domain Ω of the geometric description to an arbitrary attribute space κ, as illustrated in Figure 2. The solution in our data sets is also given in a NURBS representation. The control points $\mathbf{a}_{i,j,k} \in \kappa$ for each attribute are attached to the control points of the geometric description and are likewise evaluated.

At this point, we would like to clarify the terminology which is used in this paper to denote the components of such a volumetric representation. In this work, the term model refers to the entire mesh of trivariate NURBS solids, including the solution of the isogeometric analysis. This model can be decomposed into an equivalent representation that consists of rational Bézier solids, which we refer to as Bézier cells or just cells. The solution attached to each cell maps from the domain of the cell to attribute space and is therefore referred to as an attribute function. The boundaries of a Bézier cell are tensor product Bézier surfaces, which we refer to as faces. In particular, an inner boundary or transition denotes a face which is shared by two adjacent Bézier cells, while an outer boundary is part of the surface of the entire model.

3 Related Work

While there are numerous isosurface visualization techniques for grid-based volumes [17], irregular volumes [9] as well as higher-order finite elements [34] [14] [33], few rendering schemes for NURBS-based volume representations exist. A major reason, aside from the obviously high costs, is that most CAD applications do not enforce volume modeling [13] and thus little emphasis has been placed on the development of appropriate rendering techniques. However, the seminal work of Hughes et al. [12] introduced NURBS solids for the purpose of isogeometric analysis resulting in an increasing need for algorithms addressing this problem.

Common isosurface visualization approaches generate a polygonal approximation of the actual isosurface which is then employed for direct rendering. There are a variety of isosurface extraction algorithms such as marching cubes [17] for regular grid structures or the isosurface extraction algorithm for irregular volume data proposed by Cignoni et al. [9] [8]. While the method of extraction may vary by algorithm, the goal of each is to find a piecewise linear approximation which is sufficiently accurate for rendering. This task is a time-consuming preprocessing step and for volumes containing curved isosurfaces, the resulting mesh is likely to be inaccurate in some areas which causes visual artifacts. Furthermore, an isosurface extraction for a NURBS-based volume description would necessitate a resampling of the volume – a costly task with enormous storage requirements.

There are also a variety of rendering approaches for tetrahedral volume meshes. Most of them use either point-based methods [37], cell projection [25] or ray casting [35]. Visualization techniques for higher-order volumes build on similar techniques, but focus on finite element representations. A comprehensive summary of these methods was produced by Sadlo et al. [26].

In particular, Nelson et al. [21] presented a ray-tracing system for isosurface visualization of higher-order finite elements. The progression of the simulation data along the ray is approximated by a polynomial. Their approach allows for pixel-exact images based on an error budget, but the algorithm including recent GPU adaptations [22] did not reach interactive frame rates. Bock et al. [3] also use ray approximations in parameter space for their interactive visualization. They exploit curve similarities to cluster and compress the resulting set of curves – an approximate and time-consuming preprocessing step (up to several hours) we would like to avoid.

Furthermore, Meyer et al. [20] presented an approach in which isosurfaces of higher-order finite elements are visualized using a particle system. A set of particles is iteratively projected onto the isosurface and evenly distributed. The partial derivatives of each particle are used to determine its orientation in order to apply basic splatting algorithms for interactive rendering. The amount of particles necessary to sufficiently approximate the isosurface is view-dependent. An increasing number of particles will improve visual quality. However, splat-based methods are inefficient in generating pixel-accurate results as they are achieved through ray casting techniques.

Üffinger et al. [33] presented a rendering system which employs GPU-based ray casting for the visualization of curvilinear finite element cells with varying polynomial degree. A three-dimensional grid is used to intersect cells which are then sampled using a frequency-based sampling strategy. A disadvantage of their approach is that the grid cells might enclose a large amount of empty space. An alternative is to sort the higher-order elements into a min/max octree hierarchy similar to the approach presented by Knoll et al. [16]. However, in the evaluation of our algorithm we found that grid-based data structures do not allow us to focus on just those cells which contain a given isovalue or a set of isovalues. Instead of traversing a spatial data structure during runtime, our approach exploits current graphics hardware capabilities and generates and sorts the relevant cell intervals on-the-fly without the need of further preprocessing.

As previously mentioned, few rendering methods exist for the visualization of NURBS volumes. Chang et al. [7] were the first who presented a direct rendering approach. Similar to the particle-based approach for higher-order finite elements by Meyer et al. [20], visualization is accomplished by evaluating adaptively distributed point samples and splatting. The limitations of such splat-based approaches have already been pointed out. Raviv and Elber [24] precompute the effect of each scalar attached to the control points on the final image. The resulting mapping function allows for interactive rendering and volume manipulation, but the approach is limited to a fixed view direction. The rendering technique presented by Samuelčik [27] utilizes a polygonal approximation of isoparametric curves and isoparametric surfaces for volume visualization, but does not support isosurfaces.

Martin et al. [18] presented a robust isosurface visualization technique for various higher-order representations which can also handle NURBS primitives. In their approach, the volume representation is recursively subdivided until all intersections for the remaining subpatches can be determined using the Newton-Raphson method. While subdivision is a proven approach for robust identification of all ray-isosurface intersections, recursive algorithms are quite limited on current GPUs. This system is CPU-based and it is not clear how to adapt this to the GPU and how it would scale on a GPU.

4 Preprocessing

Let us briefly consider the challenges of ray casting a typical model. In general, a parametric function $\mathbf{p} = \Phi(\mathbf{u})$ maps from domain coordinates $\mathbf{u} = (u,v,w)^T$ to positions in world coordinates $\mathbf{p} = (x,y,z)^T$. Given the solution of an isogeometric analysis, for each point $\mathbf{p} \in \mathbb{R}^3$ in a cell we can find the corresponding coordinates $\mathbf{a}$ in attribute space by evaluating $\mathbf{a} = \Psi(\Phi^{-1}(\mathbf{p}))$. Thus, for any point $\mathbf{p}$ in world coordinates the inverse of the mapping function $\mathbf{u} = \Phi^{-1}(\mathbf{p})$ is needed to evaluate the corresponding attribute value. In general, the inverse mapping function is not available in an analytical form and requires a numerical solution. The inversion results in an overdetermined system of non-linear equations. In this work, we assume a bijective mapping function, which has a unique solution for all points that belong to a cell. For all practical purposes, most NURBS volumes comply with this limitation. Furthermore, it generally applies to volume models that are subject to an isogeometric analysis because multiple solutions would imply a self-overlap.

In our algorithm, isosurface intersections are found by evaluating the attribute function along the ray. However, the ray in world space does not map to a line segment in the domain. Instead, it is a curve of very high polynomial degree or not even parameterizable by a polynomial description, as indicated in Figure 2(a). As a consequence, the evaluation along the ray requires to find either an approximation of this curve or to repeatedly find a solution for the inverse mapping function. In this work, we choose the latter option and proceed in world coordinates. We repeatedly solve the inverse mapping function using an iterative method as described in Section 5 which works quite well due to the inherent smoothness of NURBS.

The central task that needs to be accomplished during rendering is to find intersections between the ray and implicitly defined isosurfaces in the model. As there is no analytical solution to this problem iterative sampling techniques are used to determine the actual intersection points. However, sampling requires the frequent evaluation of the trivariate data representation and is thus an expensive operation. Therefore, the main objective of our preprocessing stage is to simplify this task and thereby reduce the sampling costs at runtime.

During the first step, the NURBS representation is converted into a rational Bézier representation. The conversion is applied to the geometry as well as to the attached attributes. The standard technique of knot insertion [5] is used to perform this task. The adjacency information is kept to provide for an easy transition between neighboring Bézier cells during ray traversal. In addition, we extract the face representation of each element. The boundary of a trivariate Bézier cell consists of the set of isoparametric surfaces given by the limits of its domain ($u,v,w \in \{0,1\}$). Thus, its faces are defined by the corresponding slices of the cell's control point net. The resulting rational Bézier representation is equivalent to the NURBS-based representation, but as we will show, some of its properties simplify the ray casting algorithm with respect to the following aspects:

- The recurrent task of evaluation can be performed more efficiently on a Bézier representation than on a NURBS representation. Sederberg [30] proposed a scheme based on the Horner algorithm in Bernstein basis which allows for the evaluation of a rational tensor product Bézier surface of degree n in $O(n^2)$. The method is well suited to be used on the GPU due to its fixed register usage. We adapted this scheme for the trivariate case of degree n which results in a complexity of $O(n^3)$.

- The conversion of the model results in a set of Bézier cells for each of which we determine the respective range in attribute space. At runtime, an isosurface intersection test is performed

only if the given isovalue is in the attribute range of the respective cell. Thus, instead of sampling through the entire model, we only sample through the subset of cells which potentially contain an isosurface.

- A common technique in GPU-based ray casting is to render a conservative proxy geometry and use the resulting pixel candidates for ray generation [6]. We follow this approach and employ the convex hulls of the cells' faces, which are each generated from their Bézier control points using the QuickHull algorithm [1]. In general, this set of convex hulls is tighter to the actual boundary than a single convex hull of the NURBS representation. Thus, its projection covers fewer pixels which in turn means that fewer rays process the cell unnecessarily.

After preprocessing the parametric description of the cells, the corresponding attribute bounds, the adjacency information and the proxy geometry are then uploaded to the GPU for rendering.

5 Point Evaluation

Given a Bézier cell $\mathbf{C}(\mathbf{u})$, we find any isosurface inside it by sampling along the ray. For this, we need the ability to evaluate the attribute function for any point $\mathbf{p}$ in world coordinates — a task we refer to as point evaluation. Each point evaluation consists of the following subtasks:

1. Find the corresponding coordinate $\mathbf{u} = (u,v,w)^T$ in the domain by solving the inverse mapping function $\mathbf{u} = \Phi^{-1}(\mathbf{p})$
2. Evaluate the attribute function $\Psi(\mathbf{u})$

At first, we transform the subtask of solving the inverse mapping function for $\mathbf{p}$ into a root-finding problem. Any given point $\mathbf{p}$ on the ray $\mathbf{r}$ can be expressed as an intersection of three orthogonal planes $\mathbf{E}_0, \mathbf{E}_1$ and $\mathbf{E}_2$ in Hessian normal form

$$\mathbf{E}_j : \hat{\mathbf{n}}_j \cdot \mathbf{d}_j = 0 \tag{2}$$

The point $\mathbf{p}$ is equivalent to a point $\mathbf{C}(\mathbf{u})$ in the cell. We iteratively find the domain coordinates $\mathbf{u}_i \approx \mathbf{u}$ using Newton's method. Each step of the iteration is defined by

$$\mathbf{u}_{i+1} = \mathbf{u}_i - J^{-1}\mathbf{f} \tag{3}$$

with the remaining distance $\mathbf{f}$ to the point, given by

$$\mathbf{f} = \begin{pmatrix} \mathbf{C}(\mathbf{u}_i)\cdot\hat{\mathbf{n}}_0 + \mathbf{d}_0, \\ \mathbf{C}(\mathbf{u}_i)\cdot\hat{\mathbf{n}}_1 + \mathbf{d}_1, \\ \mathbf{C}(\mathbf{u}_i)\cdot\hat{\mathbf{n}}_2 + \mathbf{d}_2 \end{pmatrix} \tag{4}$$

and the Jacobian:

$$J = \begin{pmatrix} \frac{\partial \mathbf{C}}{\partial u}\cdot\hat{\mathbf{n}}_0 & \frac{\partial \mathbf{C}}{\partial v}\cdot\hat{\mathbf{n}}_0 & \frac{\partial \mathbf{C}}{\partial w}\cdot\hat{\mathbf{n}}_0 \\ \frac{\partial \mathbf{C}}{\partial u}\cdot\hat{\mathbf{n}}_1 & \frac{\partial \mathbf{C}}{\partial v}\cdot\hat{\mathbf{n}}_1 & \frac{\partial \mathbf{C}}{\partial w}\cdot\hat{\mathbf{n}}_1 \\ \frac{\partial \mathbf{C}}{\partial u}\cdot\hat{\mathbf{n}}_2 & \frac{\partial \mathbf{C}}{\partial v}\cdot\hat{\mathbf{n}}_2 & \frac{\partial \mathbf{C}}{\partial w}\cdot\hat{\mathbf{n}}_2 \end{pmatrix} \tag{5}$$

This iteration continues until $\mathbf{C}(\mathbf{u}_i)$ reaches a desired proximity $\|\mathbf{f}\| <= \varepsilon_p$ to the actual sample point $\mathbf{p}$ on the ray.

6 Ray casting Bézier Cells

Isosurface ray casting of a set of Bézier cells consists of two major tasks. First, the intersections between a ray and the cells' curved faces need to be found to determine the intervals inside the cell. Once these intervals are determined, potential isosurface intersections are searched by sampling along the ray.

Ray casting is a highly parallel technique which seems well-suited to GPU computation; however, both face intersection and sampling remain very expensive operations. Üffinger et al. [33] reduce these costs by employing a regular grid containing parallelepipeds as bounding volumes for the respective faces. For comparison purposes, we implemented their approach as well as an octree-based acceleration data structure. We found that such acceleration data structures perform slower than the approach suggested in this paper. The reasons for this are discussed in Section 8.

```
// first pass (Section 6.1)
for all proxy geometries {
  generate ray intervals of face intersections
}

// second pass (Section 6.2)
for all rays {
  sort list of intervals in ascending depth order
}

// third pass
for all rays {
  for all intervals of each ray {
    find face intersections          // Section 6.3
    classify ray segment             // Section 6.4
    if segment is inside cell {
      search isosurface intersections// Section 6.5
    }
  }
}
```

Fig. 3. Pseudo code of our algorithm.

Our considerations led us to the design of a GPU-based three-pass algorithm which can be summarized as follows. In the first phase of our algorithm, we generate a list of intervals for each ray. Each interval limits the range of potential intersections between the ray and the cell's face. A detailed description of this stage is given in Section 6.1. In the next step, the lists are sorted in ascending depth order to enable early ray termination. Once the lists of intervals are sorted, we process them in front-to-back order to find the face intersections (see Section 6.3). Two consecutive intersections with the same cell limit a ray segment which is classified with respect to the cell (see Section 6.4). If the ray segment is inside the cell, the intersections with isosurfaces are searched by sampling along the ray. An adaptive sampling strategy (see Section 6.5) increases the rate of convergence and thus reduces costs for sampling. The pseudo code explanation in Figure 3 summarizes the main tasks of each pass. Figure 4 illustrates an example and shows the corresponding results of each task.

6.1 Generate Ray-Interval Lists

In the first pass of our algorithm, we find the ray intervals which potentially contain face intersections. These intervals are generated by the rasterization of the convex hulls of the cells' faces. However, hulls are culled based on the cell's properties in order to minimize the number of generated intervals.

6.1.1 Culling

The convex hull of a cell's face is only rendered if it complies with one of the following requirements: 1) The face is part of the outer boundary of the model or 2) the attribute bounds of the associated cells include the current isovalue. Otherwise, the entire convex hull is discarded at the geometry processing stage of this pass. This optimization increases the performance of our approach because intervals are only generated and stored in the per-pixel lists if they potentially contribute either to the visualization of the model's boundary or reference a cell which potentially contains an isosurface.

The costs for vertex and geometry processing directly relate to the number of Bézier cells that need to be processed. For large models, a simple acceleration data structure is used to prevent this stage from becoming a bottleneck. In this data structure, the proxy geometries are organized in a small number of bins, each bin corresponding to a particular attribute range. These ranges are determined by dividing the

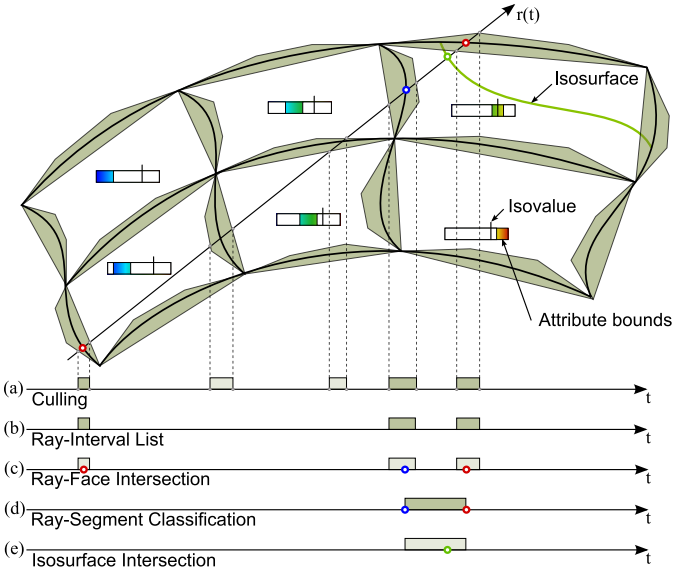


Fig. 4. The intersection between a ray and a single NURBS volume, which was converted into six Bézier cells. The ray-interval list (b) is generated from the projection of the faces' convex hulls (a). In this case, two of the hulls are culled with respect to their attribute bounds. The face intersections (c) are found by processing the ray-interval list. The ray segments in the respective cells (d) are generated from the face intersections and are searched for an isosurface intersection (e).

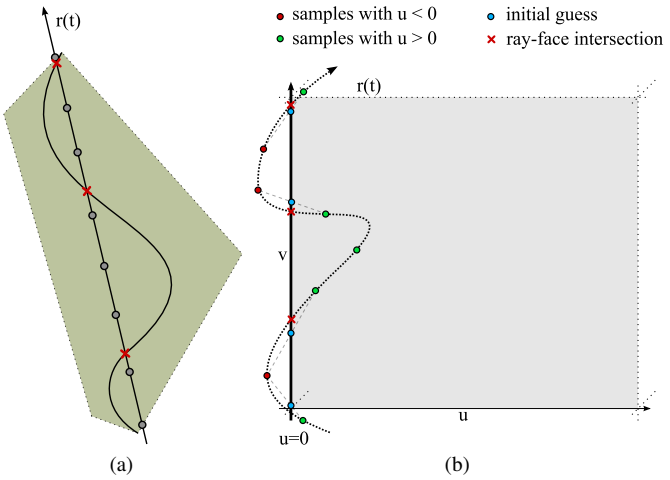


Fig. 5. This Figure illustrates our sampling-based root isolation (for clarity in a simplified 2D domain). For each sample on the ray, shown as grey points in (a), we compute the corresponding domain coordinates, which are shown as red and green points in (b). If two consecutive samples are on different sides of the domain boundary, we use the interpolated intersection (shown as blue points) as an initial guess for Newton's method.

attribute bounds of the entire model into a set of subranges. For each of these subranges, all the cells with an overlapping attribute range are determined, and the associated proxy geometries are then inserted into the corresponding bin. At runtime, only the geometry in the bin containing the current isovalue and the convex hulls of outer faces have to be rendered.

6.1.2 Generation

For each hull the rasterization results in a number of pixel candidates, also called fragments. For a single pixel, the fragments corresponding to the frontface and the backface of the convex hull form an interval which is a conservative bound of potential intersections with the face. Note that we do not intersect the faces during fragment processing. Instead, we store each fragment, as described in Section 7, with its associated values and defer further computations. At the end of this pass, for each pixel we have an unsorted list of intervals which enclose the potential intersections with the cells' faces. Figure 4 shows the generation of these intervals, denoted by (b), for a single ray. These lists are sorted in the next stage of our algorithm and later sequentially processed along the ray.

Although rendering any bounding volume would also serve for ray generation, using the convex hull has considerable advantages. In general, a convex hull is a tighter bounding volume than, for example, a parallelepiped. Consequently, its screen projection generates a lower number of fragments and thus fewer rays need to be processed. In addition, they provide a coarse surface approximation which can be exploited to generate a good initial guess for Newton's method as proposed by Pabst et al. [23].

6.2 Sort Ray-Interval Lists

Once the ray-interval lists have been constructed, every list has to be sorted in ascending depth order. Most sorting algorithms could be used to accomplish this task, but current GPUs are still limited with respect to recursions and dynamic memory allocation. We have found that Bubblesort works best within these limitations because of the following reasons: First of all, the intervals are stored in singly-linked lists which prevents the use of sorting algorithms that require random access. Furthermore, it is a non-recursive sorting algorithm which performs all memory operations in place and thus has no additional storage needs. Finally, the lists need to be sorted per pixel and the number of list elements is relatively small and varies per pixel which makes GPU-optimized sorting algorithms (such as radix sort) inefficient.

6.3 Ray-Face Intersection

The main objective of this stage is to identify the boundaries of the ray segments which are analyzed for isosurface intersections, as illustrated in (d) in Figure 4. Such a ray segment is limited by two consecutive face intersections, which are shown in (c) in Figure 4. The sequence of face intersections along the ray defines a number of mutually disjoint segments because the cells do not overlap which is a precondition to isogeometric analysis. This allows us to compute these segments step by step in ascending depth order and if necessary, analyze them for isosurfaces.

There is no closed-form solution for a curved face intersection. In general, subdivision approaches [29], interval arithmetic [15] or iterative methods [19] are used instead. The latter represent the most inexpensive approach, while the other two are more robust. In this work, we chose an iterative approach because it enables us to achieve interactive frame rates. However, we aim for a similar robustness and therefore combine the iterative approach with a sampling-based root-isolation technique.

A ray-face intersection results in a system of nonlinear equations. In order to find all the solutions, it is necessary to isolate the roots before using an iterative root-finding method. Our method is capable of isolating all face intersections along the ray within the accuracy of our heuristic, and it provides close start values for initiating Newton's method. Most other interactive rendering approaches [33] [23] [19] [28] sidestep the issue of root isolation and assume at most two intersections with a single face. We also implemented such a common intersection heuristic and compare the results of both approaches in our discussion in Section 8.

The main idea of our root-isolation technique is to exploit the relationship between a face and the trivariate domain space of the cell. Each boundary of the cell's domain defines an isoparametric surface which is equivalent to the respective face. Consequently, we need to find the roots of the signed distance function which is defined by the ray's representation in domain space and a domain boundary. The ray's representation in the domain is not known and cannot be com-

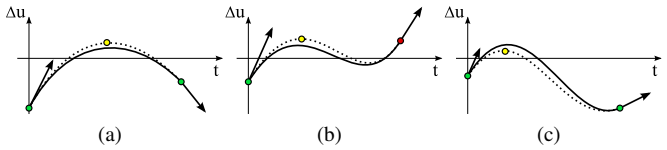


Fig. 6. These three examples illustrate the cubic interpolation (shown as dotted line) of the signed-distance function between two consecutive samples. If there is a single extremum (a) or even two extrema between the two samples, as shown in (b) and (c), we continue sampling at the first of the approximated extrema, shown as yellow points, in order to reduce the chance of missing multiple roots.

puted on-the-fly. However, two points on the ray whose distances differ in sign imply a real root. Thus, roots can be isolated by sampling along the ray, as shown in Figure 5. The interval which is sampled is limited by the ray's entry point and exit point into the convex hull. Note that sampling is not used to compute the roots, but to isolate them and provide a close initial guess for Newton' method.

Nevertheless, each point evaluation as described in Section 5 is an expensive task, and the number of samples needs to be minimized. Sampling is started at the entry point into the convex hull. After each point evaluation, we transform the ray direction into the domain in order to estimate the next face intersection using a linear extrapolation. The position of the next sample is then set to a small offset after the estimated intersection. This is because we only need to find the sign-change. As offset we use the projected screen-space error. In case of a sign-change, we approximate the intersection by means of linear interpolation, as shown in Figure 5(b), and we use the result as an initial guess for Newton's method. If the distance between entry and exit point is smaller than the size of a pixel, we assume a top view of a flat convex hull, which contains at most a single intersection. In this case, no point evaluation is required. Instead, the mean of the initial guesses provided by the two fragments is used.

In general, a linear extrapolation of the ray in domain space is associated with an error. Therefore, we provide two additional measures which both contribute to the robustness of our approach:

- The step length is limited if the distance to the estimated intersection is too large or negative. This reduces the chance of missing multiple intersections. The choice for this maximum step length depends on the current view and the curvature of the face and will be part of our discussion in Section 8.

- We adopt the idea of using a ray approximation in domain space, as shown by Bock et al. [4]. While in their approach rays are approximated for the entire domain in a preprocessing step, we use local interpolations which are computed on-the-fly. For each pair of consecutive samples, we compute a cubic interpolation of the ray-face distance and compute its extrema using its derivative, as shown in Figure 6. If this approximation has a local extremum between the two samples and the extremum indicates a sign-change, roots may have been missed. In this case, we continue sampling at the estimated position of the extremum and thereby isolate the roots which would otherwise have been missed. Furthermore, this approach also increases the quality of the interpolated initial guesses, because the signed distance function between two samples is likely to be monotonic.

Our sampling-based root-isolation approach requires performing point evaluations outside the actual cell. The bijective mapping inside the cell does not necessarily apply to the outside which may cause ambiguities. However, ambiguities are rare due to the smoothness of the polynomial basis and the fact that all such point evaluations are in close proximity of the cell's boundary. Furthermore, they do not represent a problem because the corresponding domain points can still be classified outside.

6.4 Ray-Segment Classification

Once a face intersection is found, the segment between the last two found intersections is classified as inside or outside the cell. In general, this can easily be accomplished by tracking the current state of the ray, similar to the point-in-polygon algorithm [32]. However, in our case this task might involve issues associated with the use of numerical methods such as double or missed intersections. Thus, we additionally use the cells' adjacency information which was gathered during preprocessing to resolve inconsistencies.

The adjacency information identifies whether the face belongs to the outer boundary of the model or if it represents an inner transition between two adjacent cells. An intersection with the outer boundary can be classified as an entry or an exit point, depending on the current ray state. Thus, if the ray hits an outer boundary the current state of the ray switches from outside to inside and vice versa. In contrast, an intersection with a transition face implies that the ray is inside and remains inside the model.

If the ray segment is inside the cell and the isovalue is in the corresponding attribute bounds, as indicated for the segment (d) in Figure 4, we proceed with searching for isosurface intersections by sampling through the cell.

6.5 Isosurface Intersection

Once a cell has been identified as having isosurface intersections, the next stage is to sample along the ray, using the point evaluation techniques described in Section 5. A naïve implementation would perform equidistant point evaluations along the ray and report if a transition of the respective isovalue occurs. Once a transition is found, an iterative process could sample the corresponding range to determine the actual intersection point. However, this approach would be too expensive in our case due to the large number of expensive point evaluations.

Our idea is to estimate the distance to the nearest isosurface intersection along the ray using the derivative of the attribute function in ray direction. This adaptive sampling approach allows for dense sampling of regions which are close to the isosurface intersection and skipping of parts of the cell which are quite distant from the isosurface. In the following, we derive the size of the adaptive sampling step.

For the sake of simplicity, we assume that we are dealing with scalar attributes. For higher-dimensional attributes a mapping to a scalar value would have to be provided. Based on this assumption an isosurface, which corresponds to an isovalue $\rho_0 \in \mathbb{R}$, is defined by all $\mathbf{u}$ which satisfy the equation $\rho_0 = \Psi(\mathbf{u})$. Thus, we find all intersections between the ray $\mathbf{r}(t) = \mathbf{r}_0 + t \cdot \hat{\mathbf{d}}$ and the isosurface by solving

$$0 = \rho_0 - \Psi(\Phi^{-1}(\mathbf{r}(t))). \tag{6}$$

Given a cell $\mathbf{C}$ and a point $\mathbf{s} \approx \mathbf{C}(\mathbf{u})$ on the ray, we determine the position $\mathbf{s}'$ of the next sample as follows

$$\mathbf{s}' = \mathbf{s} + \Delta_t \cdot \hat{\mathbf{d}} \tag{7}$$

where $\hat{\mathbf{d}}$ is the normalized ray direction and Δ_t denotes the distance to the next sample point.

$$\Delta_t = \begin{cases} max(t_{min}, \delta) & if\ 0 \leq \delta < t_{max} \\ t_{max} & else \end{cases} \tag{8}$$

The distance Δ_t depends on an estimate of the nearest isosurface intersection, denoted by δ. It is provided by the linear extrapolation of the attribute function along the ray. If the extrapolation diverges ($\delta < 0$) from the isosurface or the sample point is already close to the isosurface, we use an alternative distance of $t_{min/max}$ instead. The choice of these parameters is discussed in detail at the end of this section. The estimate is determined by

$$\delta = \frac{\rho_0 - \Psi(\mathbf{u})}{\frac{\partial \Psi}{\partial t}} \tag{9}$$

and the extrapolation is given by projecting the derivatives of the attribute function and the domain coordinates onto the ray:

$$\frac{\partial \Psi}{\partial t} = \frac{\partial \mathbf{u}}{\partial t} \cdot \delta_{\mathbf{u}} \tag{10}$$

$$\frac{\partial \mathbf{u}}{\partial t} = J^{-1} \cdot \hat{\mathbf{d}} \tag{11}$$

$$\delta_{\mathbf{u}} = (\frac{\partial \Psi}{\partial u}, \frac{\partial \Psi}{\partial v}, \frac{\partial \Psi}{\partial w})^T \tag{12}$$

$$J = \begin{pmatrix} \frac{\partial \mathbf{C}(\mathbf{u})}{\partial u} & \frac{\partial \mathbf{C}(\mathbf{u})}{\partial v} & \frac{\partial \mathbf{C}(\mathbf{u})}{\partial w} \end{pmatrix} \tag{13}$$

If the interval of two attribute values of two consecutive samples contains the isovalue ρ_0, a root-finding method is applied to the corresponding interval to find the exact intersection point. We have found that a bisection method works well due to its numerical stability and the proximity to the root. In order to avoid missing thin features, we also consider the values and derivatives of the attribute function along the ray as described for the face intersection in Section 6.3.

Once the isosurface intersection is found, we perform the shading operations. If we are dealing with semitransparent isosurfaces we reset the ray's origin to the point of intersection. The adaptive sampling continues until it reaches the cell's exit point or the pixel's color is saturated.

The behavior of our adaptive sampling strategy can be controlled by the parameters t_{min} and t_{max}, which form an interval which is used to clamp the distance to the next sample.

In the proximity of an isosurface, the adaptive sample distance becomes very small because it is based on an estimate of the actual intersection. In cases when we seek all isosurface intersections in sequential order, the parameter t_{min} is used as a lower limit for the distance to jump beyond the intersection found and continue the search for the remaining interval. Thus, the value t_{min} is an error bound for the minimal distance between two consecutive isosurface intersections determined by our algorithm. This parameter is set to the size of one pixel.

The alternative step size t_{max} is mainly motivated by the fact that the sample distance is based on a first order extrapolation which may be divergent or singular even if an intersection exists. This is generally caused by a local extremum. A higher-order extrapolation could be employed, but it would not guarantee convergence. However, our experience shows that $t_{max} = \frac{S}{2n}$ is a reasonable threshold, where S denotes the size of the cell and n its polynomial degree. A comparison between different choices of t_{max} is shown in Figure 12 and discussed in Section 8.

7 Efficient Per-Pixel Lists

At this point, we have to point out that an efficient implementation of the ray-interval list generation is necessary to prevent this stage from becoming a serious bottleneck. Recall that in the fragment processing stage of the first pass as described in Section 6.1.2, we store all information that is necessary to perform the face intersection at a later stage. Due to recent developments in modern graphics processors [31], it is possible to perform atomic memory operations to arbitrary locations. This scatter capability allows us to store the dynamic information in texture memory. The data structure we employ is based on the dynamically constructed linked list structure presented by Yang et al. [36]. In their approach, the concurrent access to the address of the next list element is handled using a single atomic counter. As fragments are processed by concurrent threads, the contention caused by all threads trying to update the same memory location represents a major bottleneck. They sidestep this issue by utilizing append buffers — a feature only available for DirectX 11. However, we need to remain platform-independent and have developed an allocation scheme which alleviates the effects of memory contention.

For each pixel, all fragments are stored in a singly-linked list. A list element contains the core information of a fragment: depth, face index, predecessor and the interpolation of the initial guess for the face intersection.

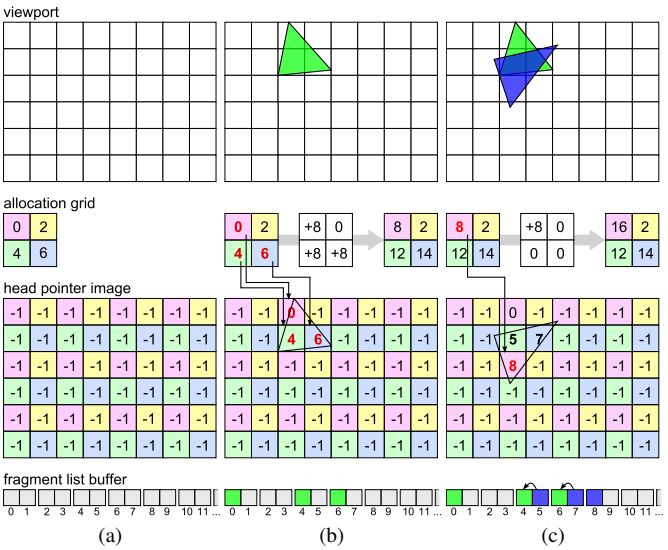


Fig. 7. In this example, two triangles (green, blue) are rendered sequentially into the per-pixel lists using an allocation grid of 2×2 and a page size of 2. The pixel colors of the head pointer image indicate the corresponding allocation counter. Initially, no pages are reserved (a). The fragments of the green triangle have to request new pages for the corresponding pixels from the allocation grid. These requests are addressed to different atomic counters and do not stall each other (b). For the blue triangle, two fragments fit into the reserved pages and only one fragment has to request a new page (c).

Figure 7 illustrates the construction of our linked-list data structure. The address of the last list element is stored in the head pointer image. All list elements are stored in a single buffer which is organized in pages. This buffer is also referred to as fragment list buffer. Every page reserves memory for a fixed number of list elements. The allocation of pages is controlled by the allocation grid. This grid is a fixed size image which holds indices to empty pages in the fragment list buffer. All indices in the allocation grid, also referred to as allocation counters, work independently from each other.

At runtime, each fragment requests the head pointer for the pixel. If there is no space left in the current page, a new page is requested from the allocation grid. The grid position which the request is addressed to is determined using modulo operations. It corresponds to the remainder of the division between the fragment's screen position and the grid resolution. At the grid position, we look up the address of an empty page and perform an atomic increase on the corresponding counter. The offset which is added depends on the page size and resolution of the grid. For a grid size of $s_x \times s_y$ and a page size of s_p, the offset is $s_x \times s_y \times s_p$. The result of this operation is the address of the next empty page owned by this counter.

The size of the allocation grid directly affects the performance and memory consumption of our algorithm. In general, a higher grid resolution improves render performance, but also implies a slight memory overhead. In the following, we elaborate the reasons for both.

The performance gain is due to the lower number of threads affected by contention. In practice, fragment programs run on blocks of fragments. Each fragment is processed concurrently and in the worst case, all threads request an empty page from the allocation grid at the same time. However, adjacent fragments address different allocation counters due to the applied modulo operations. As all threads in the same block process neighboring fragments, concurrent access to the same counter is rare. Furthermore, if the grid resolution is higher than the block size, all threads in one block address different counters. Thus,

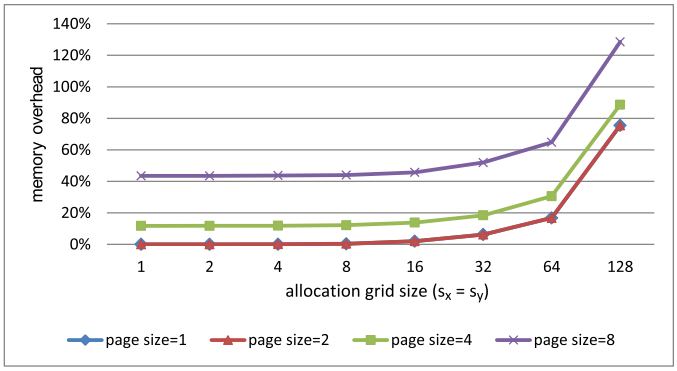


Fig. 8. The memory overhead of the linked list generation for various configurations. Using a page size of 4 and an allocation grid of 64×64, the list requires about 25% more memory for unused list entries.

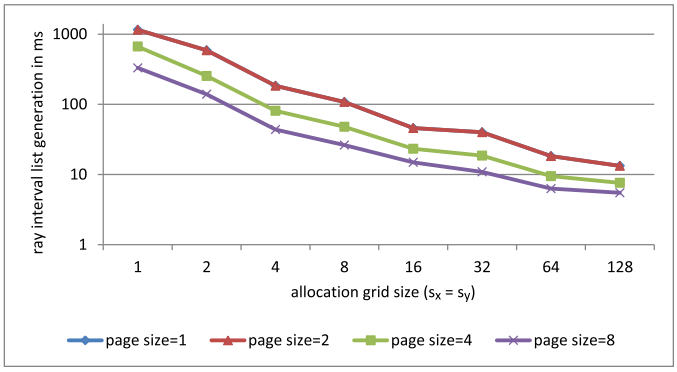


Fig. 9. Costs for the fragment list generation for the close-up on the bridge model shown in Figure 1(c) in various configurations. The performance scales with both page size and allocation grid size. A maximum performance of about 5.5 ms for more than 4 million fragments is reached using an allocation grid of 128x128 and a page size of 8.

increasing the grid resolution minimizes the number of concurrent accesses to the same atomic counter and improves performance.

The downside of a higher grid resolution is an increasing memory overhead. Each allocation counter separately owns a number of pages. With respect to ownership, all pages in the fragment list buffer are interleaved. A page request returns the next page for that counter even though some pages in between may remain unused. In the example shown in Figure 7, the second page remains empty because no request is addressed to the corresponding yellow counter. However, pixels that access the same allocation counter are evenly distributed across the screen and the number of page requests for each counter is similarly high. An evaluation of various configurations, as shown in Figure 8, indicates that the memory overhead remains fairly low in the range of about 15% to 60% for a grid size of 64×64. Figure 9 shows that the corresponding performance gain reaches up to a factor of 100x.

8 Results and Discussion

Figure 10 illustrates the technical realization of our three-pass algorithm, which was implemented as described. The first pass is implemented using OpenGL and GLSL as it depends on the rasterization capabilities. The resulting fragments are stored using the image load and store capabilities [31]. Subsequently, the sorting pass and the ray casting are performed by compute kernels which are implemented in CUDA. However, the CUDA API does not provide the capability to directly write into the framebuffer. An off-screen render target is used instead and finally mapped to the screen.

All tests were performed on a 3.33 GHz Intel Core i7 workstation with 12GiB RAM equipped with a single NVIDIA Geforce GTX Titan graphics board with 6GiB video memory and using a window resolution of 1024x1024. The linked list generation was configured using a page size of 4 and an allocation grid of 64x64.

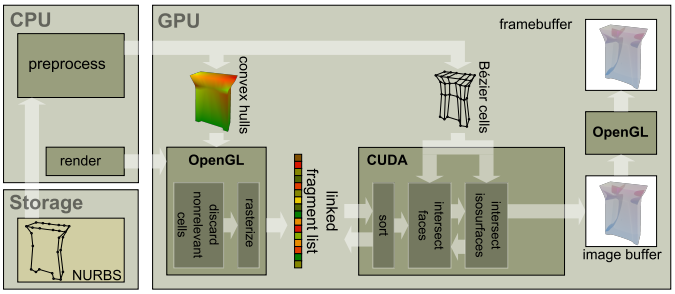


Fig. 10. A schematic overview of our multi-pass ray casting system.

	NURBS		Bézier representation		
	solids	#points	solids	#points	degree
bridge	16	40,096	6296	169,992	$2 \times 2 \times 2$
wind simulation	120	789,680	705,550	19,049,850	$2 \times 2 \times 2$

Table 1. The bridge data set comprises the following attributes: engineering stress, von Mises stress, engineering strain, and displacement. The wind simulation comprises displacement, velocity and acceleration.

We evaluated our approach using two datasets: A NURBS-based isogeometric analysis applied on a bridge model, as shown in Figure 1; and a NURBS-based wind simulation of the air flow around one rotor of a wind turbine as shown in Figure 11. The initial preprocessing varies from less than a second for the bridge model to a few seconds for the wind simulation. Table 1 shows the resulting number of Bézier cells for each model. The rendering performance is mainly fill-rate dependent; it varies between 9 Hz to 18 Hz for the views shown in Figure 1 and is about 6 Hz for the view shown in Figure 11. The respective timings for the three passes of our algorithm are given in Table 2. The rendering performance for the view shown in Figure 1(b) is slightly better considering that it benefits from higher cache coherence and lower depth complexity.

First, we would like to discuss the memory requirements of our algorithm. The main idea to focus the search for isosurfaces on the relevant parts of the model requires a conversion from NURBS to a rational Bézier representation. While this is necessary to determine the geometric and attribute bounds for the respective knot spans, it also increases the number of control points, as shown in Table 1, and thus memory requirements. The resulting overhead is of an order $(n+1)^3$, where n is the polynomial degree. Besides the parametric representation, a polygonal description of the faces' convex hulls is required including a per-vertex attribute for the initial guess. In our current implementation, the Bézier control points are kept and used for efficient evaluation. This memory overhead could be reduced by keeping only the proxy geometry and the associated knot span for a direct evaluation of the NURBS representation. However, additional memory is also used for the generation of the ray-interval lists. The budget reserved for this data structure is mainly depth-complexity and resolution dependent. In our current implementation, storing the necessary information for a single fragment requires 16 bytes and we found that a budget of 256MiB of graphics card memory was sufficient even for views with a high depth complexity.

At this point, we would like to discuss the choice of Newton's method for numerical root finding and also point out the associated limitations with respect to our algorithm. The convergence properties, advantages and issues of this method are thoroughly discussed in [11]. In our algorithm, Newton's method is used for four purposes: root isolation, ray-face intersection, point evaluation and as a heuristic for adaptive sampling. The latter increases rendering performance, as shown in Figure 16, and convergence issues are addressed by limiting the resulting sampling distance with satisfying results as shown in Figure 12. Likewise, we set the maximum step length of our sampling-

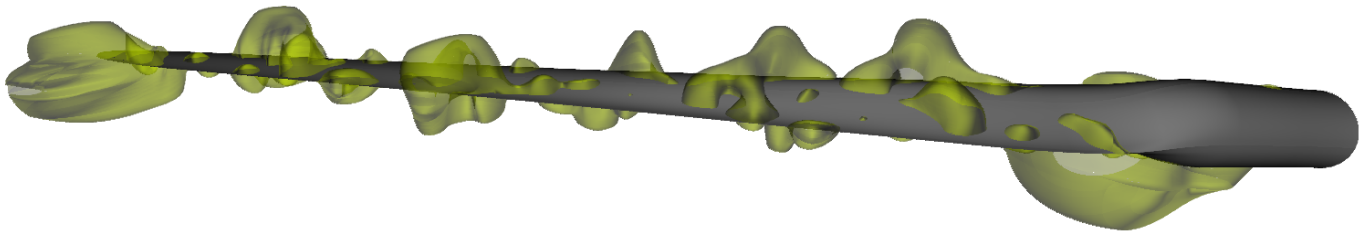
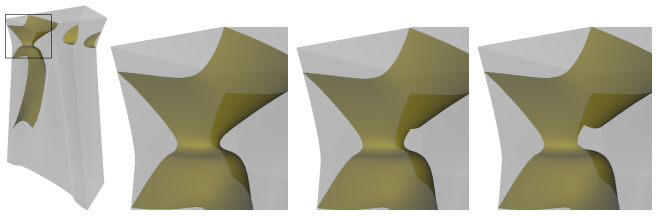


Fig. 11. This Figure shows the air displacement around the rotor of a windturbine. The rotor is a separate object which consists only of NURBS surfaces and is rendered using conventional NURBS ray casting [23] in a separate rendering pass. Some of the isosurfaces may not appear entirely smooth. This is due to the C_0-continuity at the boundary of adjacent NURBS elements and not an artifact of the proposed rendering algorithm.

view	#fragments	list generation	sorting	ray casting
bridge	824,858	2.6ms	3.2ms	49ms
pier	1,677,022	4.1ms	2.9ms	42ms
close-up	4,025,634	9.5ms	5.6ms	97ms
wind simulation	6,523,424	26.1ms	10.2ms	129ms

Table 2. This Table provides a detailed overview of the rendering times for views shown in Figures 1 and 11. The view of the pier (Fig. 1(b)) runs at 18 Hz and benefits from discarding inner cells which do not contain an isosurface. In contrast, the close-up view (Fig. 1(c)) renders at about 9Hz due to a higher fill-rate and a larger number of cells in which a search for an isosurface is necessary. The list generation for the view of the wind simulation (Fig. 11) is more expensive due to larger number of processed cells.



(a) (b) $t_{max} = \frac{S}{2n}$ (c) $t_{max} = \frac{S}{n}$ (d) $t_{max} = \infty$

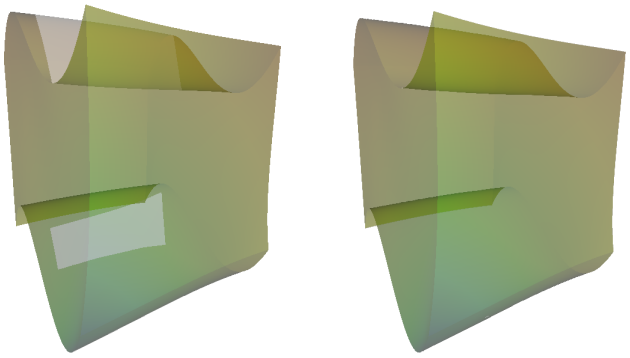
Fig. 12. The Figures (b) to (d) show the visual quality for different choices of t_{max}. A close-up on the isosurface visualization shown in (a) reveals that a higher maximum sampling distance may cause visual artifacts in regions with high curvature.

based root isolation approach to S/n, where S is the size of the interval which is sampled and n the degree of the face. This choice worked well for all our models. However, adapting these parameters to the curvature, the current view and the error tolerance is desirable, but also quite involved and remains future work. The point evaluation usually converges to the only existing solution because the preceding sample provides a very close initial guess. The convergence rate for finding ray-face intersections benefits from our root isolation approach.

In practice, most models contain at least some degenerate cells. In our wind simulation, for example, the space around the rotor is represented by a set of tubular sections. The inner faces at the center of the tube degenerate to single edges. These degenerate cells are located around the base of the rotor shown in Figure 11. In general, most degeneracies are a result of one or more collapsed edges. During preprocessing, we detect all faces which degenerated into a single edge or even into a single point and exclude them from the rendering process. In this case, all entry and exit points can still be found because such degenerate faces are coincident with an edge or a point of an adjacent face. However, faces with only a single collapsed edge do not undergo special treatment. As a consequence, the partial derivatives close to the collapsed edge become very small. Point evaluations in these regions are limited by machine precision and the convergence properties of Newton's method. Adapting the iteration with respect to the type of degeneracy could reduce the problem, but remains future work.

While the bijective mapping may be valid inside a cell, it does not necessarily apply to the outside. In particular degenerate cells tend to have ambiguities even close to the boundary. For our sampling-based root isolation, this does not represent a problem. If Newton's method converges to one of the multiple solutions, the corresponding point in domain space can still be classified as being outside.

Furthermore, we compared our results to the common intersection heuristic which is used by most other interactive rendering approaches [33] [23] [19] [28]. In their work, two initial guesses are generated using the intersections with a convex proxy geometry. Newton's method is then started for both, assuming not more than two intersections between the ray and the contained face. For the purpose of comparison, we implemented such a heuristic and use the interpolated initial guess which is provided by the rasterization of the textured



(a) Common heuristic (b) Our approach

Fig. 13. The visualization of the outer boundary of a tricubic Bézier volume. The upper and lower face are highly curved. For these faces, the common intersection heuristic (a), which is used by Üffinger et al., causes visual artifacts. By contrast, our approach (b) provides close initial guesses and finds all face intersections.

convex hull. This heuristic is slightly faster compared to our approach, as shown in Figure 15. For most of our models, the visual results are also the same. This is mainly because most parts of our models have a low curvature due to the refinement process of the isogeometric analysis. These parts apply the assumption of two or less intersections per face. However, parts with higher curvature are affected by this limitation which causes visual artifacts, as shown in Figure 13. For the same view, there are no visual artifacts using sampling-based root isolation.

The performance chart in Figure 15 includes the draw times for our approach as described and a variant of our approach employing the common intersection heuristic. The performance gain using the latter is relatively low because the sampling for root isolation adapts to the curvature of the face and the thickness of its convex hull. Thus, most point evaluations related to root isolation are performed in regions where it is necessary, e.g. at the silhouette of the model.

We compared our algorithm to an implementation of the approach presented by Üffinger et al. [33]. In addition, we implemented an alternative in which the parallelepipeds are organized in an octree data structure instead of a regular grid. The octree adapts better to different cell sizes and complex spatial configurations. This is very visible in the frame rate variations of Üffinger et al. To ensure comparability, both approaches use the same kernel for the actual isosurface intersection as our approach. At first glance, the detailed rendering times of our performance tests in Table 2 suggest that the construction of the ray-interval lists incurs a certain overhead in comparison to a spatial acceleration scheme. But, as a matter of fact, our algorithm performs much faster than both other approaches, as shown in Figure 15, because it takes a lot of computational load off the actual ray casting kernel by generating a lower number of rays, exploiting the GPU's rasterization capabilities to intersect the proxy geometry and providing closer initial guesses for the intersection of the cells' faces. In addition, we found that performing the major tasks in separate passes (as shown in Figure 14) is much more cache-coherent than a single-pass approach. The reasons for this are:

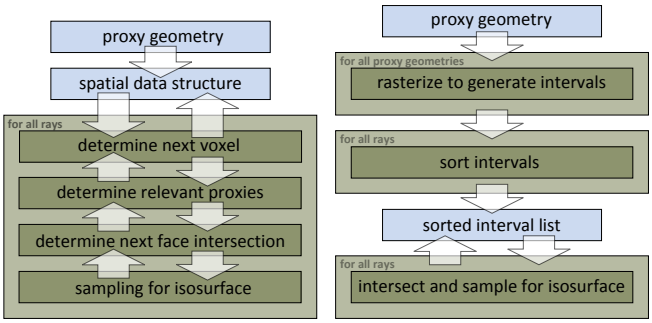
- *Segmentation*: For the purpose of analysis, the NURBS basis is typically refined in regions with high curvature. As a result, the cells' sizes differ significantly which makes using a regular grid rather unsuitable. In our tests, even grids with a very high resolution contained grid cells with more than thousand faces. Our alternative octree-based implementation easily adapts to different cell sizes. However, in both data structures it is inevitable that many faces overlap multiple nodes or grid cells which causes a considerable memory and performance overhead.

- *Cache coherence*: A modern GPU operates on blocks of threads which are processed in parallel. All threads in a single block accessing the same data benefit from the GPU's texture cache. However, the memory access can be quite incoherent for each thread processing a single ray. Using a spatial acceleration structure, the data required for ray traversal and intersection tests may be entirely different, even for adjacent rays. By checking the attribute bounds during geometry processing and using rasterization for intersecting the convex hulls we move tasks which potentially cause incoherent memory access to earlier stages of the pipeline, where they are processed more efficiently.

- *Proxy intersection*: The ray segment which is analyzed for face intersections is limited by the entry point and exit point of the face's proxy geometry. Each point provides an initial guess for the point evaluation. In general, a convex hull provides a tighter bounding volume than a parallelepiped or a bounding box. Consequently, using convex hulls results in shorter ray segments. Furthermore, the convex hulls are more suitable to attach appropriate parameters for an initial guess [23]. However, an analytical intersection of a ray and a convex hull would require many more operations than the intersection with parallelepipeds. Instead, we exploit the rasterization capabilities of the graphics hardware to perform this task.

- *Preprocessing*: Our long-term objective to integrate our visualization approach into an interactive isogeometric pipeline requires a rapid response to changes in design and the corresponding simulation. While additional data structures help to accelerate the ray casting for a static geometry, they are a potential bottleneck for interactive updates of the model. In contrast, our approach requires only minimal preprocessing and allows for partial updates of the model.

Furthermore, we evaluated the effect of the adaptive sampling strategy. Figure 16 illustrates that the number of point evaluations is highly reduced in comparison to an equidistant sampling approach. The resulting image quality was even better for the adaptive sampling approach because the minimum sampling distance t_{min} can be set much lower than the step width of the equidistant approach.



(a) Common approach using a spatial data structure (b) Our approach

Fig. 14. A ray-casting kernel using a spatial data structure (a) requires to frequent switches between different tasks. Even for adjacent rays, the path of execution may be completely different which is in most cases cache-inefficient. In our approach (b) major tasks are performed in separate passes to take full advantage of the GPU's SIMD processing capabilities.

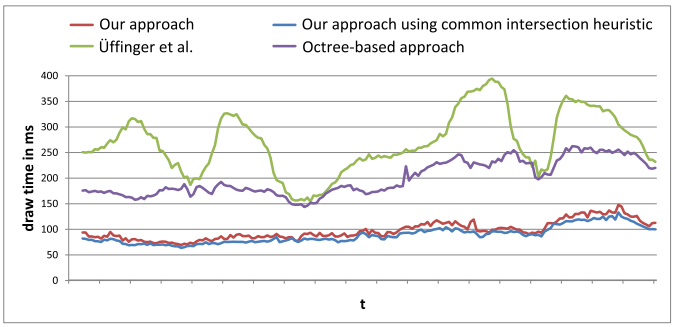


Fig. 15. This graph shows the draw times for different views of the wind simulation (shown in Figure 11) using the approach of Üffinger et al., an octree-based adaptation and our method. Our approach performs about 2 to 3 times faster than both other approaches. For this model, using our sampling-based root isolation results in about 10% slower performance compared to the common intersection heuristic.

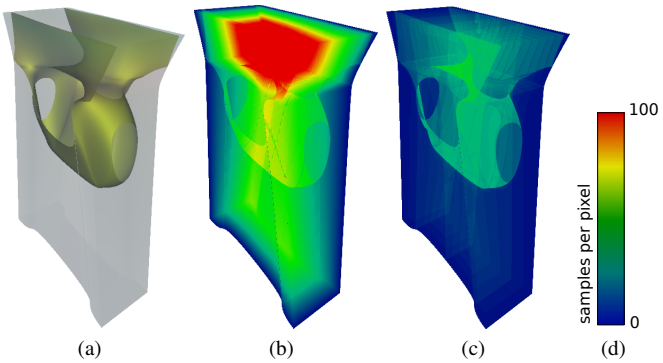


Fig. 16. This Figure shows the number of point evaluations for a view of the strain in a single pier (a). In comparison to equidistant sampling (b), adaptive sampling (c) reduces the number of point evaluations required.

Our approach sorts the ray-interval lists based on the intersection with the convex hulls of the cells' faces. In theory the order of the actual face intersections along the ray could be different. This is not a problem for the rendering of opaque isosurfaces since the traversal will continue until another intersection is found in front of the next interval. For transparent isosurfaces multiple intersections need to be blended in the right order. The computed face intersections are therefore stored temporarily in the corresponding entry of the already allocated interval lists. If there are more than two face intersections, we insert new entries in the interval list using the allocation scheme described in Section 7. Once the next interval lies behind all so far found face intersections the isosurface search is performed. Thus, early ray termination can still be efficiently used.

We would also like to point out that the applicability of our approach is not limited to isosurface rendering of NURBS-based isogeometric analysis. Of course, our approach also applies to NURBS and Bézier volumes of arbitrary degree with attached scalar fields or attribute functions. Furthermore, the main idea to generate only ray intervals requiring further computations could be adapted to other higher-order representations. In general, the ray intervals can also be used for direct volume rendering (DVR) by exchanging the search for isosurfaces with regular sampling. Using a linear approximation of the attribute function between the samples, volume pre-integration [35] may be used to accumulate color and opacity contributions. However, the performance of our approach mainly benefits from culling, adaptive sampling and a higher cache coherence by processing only a few relevant intervals. For DVR, the number of relevant cells depends on the given transfer function. In comparison to isosurface rendering, the number of generated intervals would be higher which increases storage needs, costs for sorting and the number of point evaluations.

9 Conclusion and Future Work

We developed a GPU-based ray casting system for the direct visualization of the results of a NURBS-based isogeometric analysis. Our system exploits current graphics hardware capabilities to construct ray-interval lists on-the-fly which contain all intervals that are relevant for the current isosurface visualization. Based on these interval lists, we demonstrated how to efficiently search for an isosurface directly using the underlying parametric volume and attribute representation without the need of discretization or approximation.

We support NURBS volumes of arbitrary degree, which is enabled by our adaptation of Sederberg's [30] scheme to the trivariate case. Thus, our system always generates pixel-accurate visualizations of the isosurfaces within the limitations of the numerical approaches employed. The combination of our root-isolation approach and Newton's method represents a robust, adaptive and still interactive solution for finding multiple ray-face intersections. We also provide a mathematically founded detailed description of a practical GPU-based implementation of our system which interactively visualizes models consisting of hundreds of thousands of cells with minimal preprocessing costs. Furthermore, we show how to avoid the memory allocation bottleneck for creating per-pixel linked lists and demonstrated that our approach is more than 100 times faster than a naïve implementation. A comparison to GPU-based ray casting implementations using a regular grid and an octree data structure shows that our approach results typically in a factor of 2 to 3 higher frame rates due to our efficient processing scheme.

There are many further options to refine and optimize our approach. The preprocessing could be extended with an additional refinement stage, since currently we depend directly on the refinement level of the isogeometric analysis. An adaptive subdivision of cells with a large attribute range would reduce the number and lengths of ray segments which have to be analyzed for isosurface intersections. This subdivision creates a hierarchy for each cell, which could also aid level-of-detail management for handling larger models. Additionally, the development of an appropriate occlusion culling technique is required for scenes with high depth complexity. Since we process only those cells which belong to the boundary of a model and those that contain an isosurface, the number of cell intervals per ray is typically small. However, if we were to render isosurfaces simultaneously for multiple isovalues the number of cell intervals could potentially become larger even though only the first entries are typically needed. Furthermore, the interval sorting step could significantly benefit from a coarse front-to-back rendering of the cells' surfaces.

In structural dynamics it is common to simulate dynamic models, which are represented by a sequence of discrete time steps. Our aim is to move towards a full 4D NURBS-model such that the visualization could show a smooth transition between the different time steps. Ideally the structural engineers would not only avoid the discretization of the geometry but instead generate the 4D model directly and therefore avoid the discretization in time as well. Our direct visualization approach is an important first step towards a complete isogeometric pipeline which allows for the design, simulation and visual analysis of volumetric NURBS models without the need for an intermediate approximation of the geometry.

Acknowledgments

We thank Yuri Bazilevs, University of California, San Diego, and Ido Akkerman, University of Durham, for providing the isogeometric analysis of the wind turbine and valuable feedback. Furthermore, we thank Timon Rabczuk, Michael Schwedler and Carsten Koenke at the Institute of Structural Mechanics at Bauhaus-Universität Weimar, for many helpful discussions and providing the bridge model. We also thank the reviewers for their thorough reviews and constructive feedback which significantly improved the exposition of this paper. This work was supported by the Thuringian Ministry of Education and Cultural Affairs (TKM) under grant B514-090521.

References

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22:469–483, December 1996.

[2] Y. Bazilevs, V. M. Calo, T. J. R. Hughes, and Y. Zhang. Isogeometric fluid-structure interaction: theory, algorithms, and computations. *Computational Mechanics*, 43(1):3–37, 2008.

[3] A. Bock, E. Sunden, B. Liu, B. Wnsche, and T. Ropinski. Coherency-based curve compression for high-order finite element model visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2315–2324, 2012.

[4] A. Bock, E. Sunden, B. Liu, B. Wunsche, and T. Ropinski. Coherency-based curve compression for high-order finite element model visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2315–2324, 2012.

[5] W. Boehm. Inserting new knots into B-spline curves. *Computer-Aided Design*, 12(4):199–201, July 1980.

[6] N. A. Carr, J. D. Hall, and J. C. Hart. The ray engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '02, pages 37–46, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[7] Y.-K. Chang, A. Rockwood, and Q. He. Direct rendering of freeform volumes. *Computer-Aided Design*, 27(7):553 – 558, 1995.

[8] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3:158–170, 1997.

[9] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal isosurface extraction from irregular volume data. In *Proceedings of the 1996 symposium on Volume visualization*, VVS '96, pages 31–38, Piscataway, NJ, USA, 1996. IEEE Press.

[10] J. Cottrell, T. Hughes, and A. Reali. Studies of refinement and continuity in isogeometric structural analysis. *Computer Methods in Applied Mechanics and Engineering*, 196(41-44):4160 – 4183, 2007.

[11] P. Deuflhard. *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*. Springer Series in Computational Mathematics. Springer, 2006.

[12] T. Hughes, J. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135 – 4195, 2005.

[13] Z. Jianwen, F. Lin, and S. H. Soon. A volume modeling component of CAD. In *Volume Graphics*, pages 103–117, 2001.

[14] J. Kloetzli, M. Olano, and P. Rheingans. Interactive volume isosurface rendering using bt volumes. In *SI3D*, pages 45–52, 2008.

[15] A. Knoll, Y. Hijazi, C. Hansen, I. Wald, and H. Hagen. Interactive ray tracing of arbitrary implicits with SIMD interval arithmetic. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, pages 11–18, Washington, DC, USA, 2007. IEEE Computer Society.

[16] A. Knoll, I. Wald, S. Parker, and C. Hansen. Interactive isosurface ray tracing of large octree volumes. *Symposium on Interactive Ray Tracing*, 0:115–124, 2006.

[17] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.

[18] T. Martin, E. Cohen, and M. Kirby. Direct isosurface visualization of hex-based high-order geometry and attribute representations. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):753–766, 2012.

[19] W. Martin, E. Cohen, R. Fish, and P. Shirley. Practical ray tracing of trimmed NURBS surfaces. *Journal of Graphical Tools*, 5(1):27–52, 2000.

[20] M. Meyer, B. Nelson, R. M. Kirby, and R. Whitaker. Particle systems for efficient and accurate high-order finite element visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1015–1026, 2007.

[21] B. Nelson and R. M. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):114–125, 2006.

[22] B. Nelson, E. Liu, R. M. Kirby, and R. Haimes. Elvis: A system for the accurate and interactive visualization of high-order finite element solutions. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2325–2334, 2012.

[23] H.-F. Pabst, J. Springer, A. Schollmeyer, R. Lenhardt, C. Lessig, and B. Froehlich. Ray casting of trimmed NURBS surfaces on the gpu. *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 151–160, Sept. 2006.

[24] A. Raviv and G. Elber. Interactive direct rendering of trivariate b-spline scalar functions. *IEEE Transactions on Visualization and Computer Graphics*, 7:109–119, 2001.

[25] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings of the Conference on Visualization '00*, VIS '00, pages 109–116, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[26] F. Sadlo, M. Üffinger, C. Pagot, D. Osmari, J. Comba, T. Ertl, C.-D. Munz, and D. Weiskopf. Visualization of cell-based higher-order fields. *Computing in Science and Engineering*, 13:84–91, 2011.

[27] M. Samuelčík. Visualization of trivariate NURBS volumes. *Journal of Applied Mathematics*, 2(1):143–150, 2009.

[28] A. Schollmeyer and B. Fröhlich. Direct trimming of NURBS surfaces on the gpu. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 47:1–47:9, New York, NY, USA, 2009. ACM.

[29] T. Sederberg and T. Nishita. Curve intersection using Bézier clipping. *Computer Aided Design*, 22(9):538–549, 1990.

[30] T. W. Sederberg. Point and tangent computation of tensor product rational Bézier surfaces. *Computer Aided Geometric Design*, 12(1):103–106, 1995.

[31] M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 4.2). www.opengl.org/documentation/specs, August 2011.

[32] M. Shimrat. Algorithm 112: Position of point relative to polygon. *Commun. ACM*, 5(8):434–, Aug. 1962.

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

[34] I. Wald, H. Friedrich, A. Knoll, and C. D. Hansen. Interactive isosurface ray tracing of time-varying tetrahedral volumes. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1727–1734, Nov. 2007.

[35] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 44–, Washington, DC, USA, 2003. IEEE Computer Society.

[36] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz. Real-time concurrent linked list construction on the gpu. *Computer Graphics Forum*, 29(4):1297–1304, 2010.

[37] Y. Zhou and M. Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1229–1236, Sept. 2006.

Andre Schollmeyer is a PhD candidate with the Virtual Reality Systems group at the Bauhaus-Universität Weimar. His research interests include real-time rendering techniques, GPGPU programming, ray tracing and point-based rendering.

Bernd Froehlich is chair of the Virtual Reality Systems group (`www.uni-weimar.de/medien/vr`) and a full professor with the Media Faculty at Bauhaus-Universität Weimar. His research interests include real-time rendering, visualization, 2D and 3D input devices, 3D interaction techniques, display technology, and support for collaboration in colocated and distributed virtual environments.