Direct Isosurface Visualization of Hex-Based High-Order Geometry and Attribute Representations

Tobias Martin, Elaine Cohen, and Robert M. Kirby, *Member*, *IEEE*

Abstract—In this paper, we present a novel isosurface visualization technique that guarantees the accurate visualization of isosurfaces with complex attribute data defined on (un)structured (curvi)linear hexahedral grids. Isosurfaces of high-order hexahedral-based finite element solutions on both uniform grids (including MRI and CT scans) and more complex geometry representing a domain of interest that can be rendered using our algorithm. Additionally, our technique can be used to directly visualize solutions and attributes in isogeometric analysis, an area based on trivariate high-order NURBS (Non-Uniform Rational B-splines) geometry and attribute representations for the analysis. Furthermore, our technique can be used to visualize isosurfaces of algebraic functions. Our approach combines subdivision and numerical root finding to form a robust and efficient isosurface visualization algorithm that does not miss surface features, while finding all intersections between a view frustum and desired isosurfaces. This allows the use of view-independent transparency in the rendering process. We demonstrate our technique through a straightforward CPU implementation on both complex-structured and complex-unstructured geometries with high-order simulation solutions, isosurfaces of medical data sets, and isosurfaces of algebraic functions.

Index Terms—Isosurface visualization of hex-based high-order geometry and attribute representations, numerical analysis, roots of nonlinear equations, spline and piecewise polynomial interpolation.

1 Introduction

The demand for isosurface visualization techniques arises in many fields within science and engineering. For example, it may be necessary to visualize isosurfaces of data from CT or MRI scans on structured grids or numerical simulation solutions generated over approximated geometric representations, such as deformed curvilinear high-order (un)structured grids representing an object of interest. In this context, high-order means that polynomials with degree > 1 are used as the basis to represent either the geometry or the solution of a Partial Differential Equation (PDE). High-order data are the set of coefficients for these solutions.

Given one of these representations, a visualization technique such as the Marching Cube (MC) technique [28], direct isosurface visualization [37], or surface reconstruction applied to a sampling of the isosurface is frequently used to extract the isosurface. However, given high-order data representations, we seek visualization algorithms that act natively on different representations of the data with quantifiable error.

In this paper, we present a novel and robust ray frustum-based direct isosurface visualization algorithm. The method is exact to pixel accuracy, a guarantee which is formally shown, and it can be applied to complex attribute data embedded in complex geometry. In particular, the method can be applied to the following representations:

1. Structured hexahedral (hex) geometry grids with discrete data (e.g., CT or MRI scans). The proposed method filters the discrete data with an interpolating or approximating high-order B-spline filter [29] to create a high-order representation of the function that was sampled by the grid.
2. Structured hex-based representations with high-order attribute data, where the geometry can be represented using trilinear or higher order basis.
3. Structured and unstructured hex meshes, each of which element's shape may be deformed by a mapping (curvilinear shape elements) and with simulation data (higher polynomial order).
4. Algebraic functions. The representation is exact.

We demonstrate that our method is up to three times faster and requires fewer subdivisions and, therefore, less memory than related techniques on related problems.

An added motivation to this work is the fact that trivariate NURBS [7] have been proposed for use in Isogeometric Analysis (IA) [18] to represent both geometry and simulation solutions [18], [8], [46]. Simulation parameters are specified through attribute data, and the analysis result is represented in a trivariate NURBS representation linked to the shape representation. This is the first algorithm that can produce accurate visualizations of isogeometric analysis results.

With degree >1 in each parametric direction and varying Jacobians (i.e., nonlinear mappings), trivariate NURBS that

• *The authors are with the School of Computing, University of Utah, 72 S. Central Campus Drive, Warnock Engineering Building, Salt Lake City, UT 84112. E-mail: {martin, cohen, kirby}@cs.utah.edu.*

Manuscript received 30 Mar. 2010; revised 22 Mar. 2011; accepted 6 Apr. 2011; published online 13 June 2011.
Recommended for acceptance by P. Rheingans.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2010-03-0077.
Digital Object Identifier no. 10.1109/TVCG.2011.103.

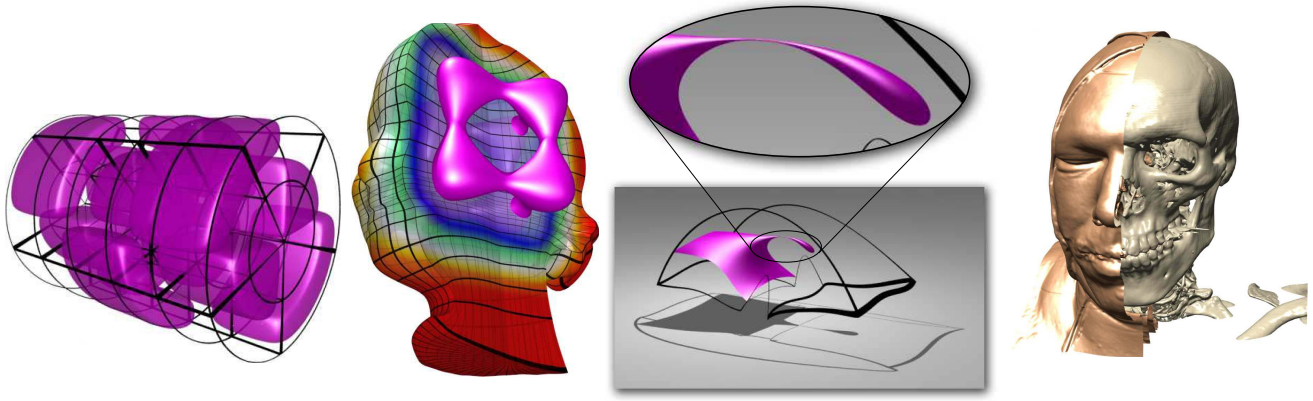

Fig. 1. Our method applied to four representative isosurface visualizations. (a) Vibration modes of a solid structure. (b) Solution to the Poisson equation. (c) Teardrop under nonlinear deformation. (d) Two isosurfaces of the visible human data set.

represent an object of interest (see Fig. 1) have no closed-form inverse. Existing visualization methods designed to work efficiently on regular spatial grids have not been extended to work robustly and efficiently and preserve smoothness on these complex and high-order geometries. Furthermore, standard approaches for direct visualization are ray based and assume single entry and exit points of a ray within an element. That hypothesis is no longer true for curvilinear elements. Hence, those approaches are difficult to extend to arbitrary complex geometry with curvilinear elements. Note that finding the complete collection of entry and exit points into curvilinear elements is a nontrivial task.

In practice, representations of more complex geometry on which numerical simulation techniques are applied often contain geometric degeneracies resulting from either mesh generation or the data-fitting process. For instance, poorly shaped elements can lead to a Jacobian with a determinant close to zero, which presents challenges during simulations. In addition, and more importantly for this paper, it presents a challenge in visualizing isosurfaces of the high-order simulation solution. Thus, there is a need for isosurface visualization techniques that deal robustly with both degenerate and near-degenerate geometry.

After discussing relevant work and the mathematical framework in Section 2, we define our mathematical formulation by stating the visualization problem in Section 3, which is solved in Section 4. Implementation details are given in Section 5, and Sections 6 and 7 analyze the results of our technique, followed by a conclusion.

2 Background

Visualization techniques are used in numerous engineering fields—including medical imaging, geosciences, and mechanical engineering—to generate a 2D view of a 3D scalar or vector data set. Additionally, they can visualize simulation results (e.g., generated with the finite element method). Consequently, the development of such visualization algorithms has received much attention in the research community. Techniques usually fall into three groups: 1) direct volume rendering, 2) isosurface mesh extraction followed by isosurface mesh rendering, and 3) direct rendering of isosurfaces.

Techniques in category 1 typically involve significant computation, especially when dealing with arbitrary geometric topologies represented by high-order basis functions such as NURBS. In ray-based direct volume rendering methods (see [26], [31]), it is necessary to integrate each ray through the volume using sufficiently many integration steps. Each integration step requires an expensive root solving due to the nonlinear mapping. Hua et al. [17] presented an algorithm to directly render attribute fields of tetrahedral-based trivariate simplex splines by integrating densities along the path of each ray corresponding to a pixel. In the case of uniform grid data sets, accumulating slices aligned along the viewing direction (see [45]) is efficient and commonly used in practice, even though ray-based techniques offer a range of optimizations (e.g., empty space skipping).

Methods in category 2 assume a regular grid of data and extract isosurfaces using MC [28], resulting in a piecewise planar approximation of the isosurface. After isosurface mesh extraction, the faces of the isosurface mesh are rendered. Marching Tetrahedra (MT) [6] is applied to both structured and unstructured tetrahedra-based grids. In both MC and MT, the corners of a hexahedral or tetrahedral element, respectively, are used to determine if the isosurface passes through the respective element. Then, the intersections between the element's edges and the isosurface are determined to create piecewise linear facets approximating the isosurface. Although these approaches are efficient and, therefore, widely used in practice, they approximate the isosurface by piecewise linear facets within an element with some ambiguity and, therefore, do not guarantee topological correctness. As an example, Fig. 2 shows the domain from Fig. 1c, represented with a single triquintic NURBS element, discretized with 300,000 tetrahedra. As seen in Fig. 2a, the respective isosurface extracted with MT has ambiguities in the topology, resulting from data that are known only at the corners of the elements and, hence, can miss isosurface features. Furthermore, the time to construct the respective mesh representation can be computationally laborious. Schreiner

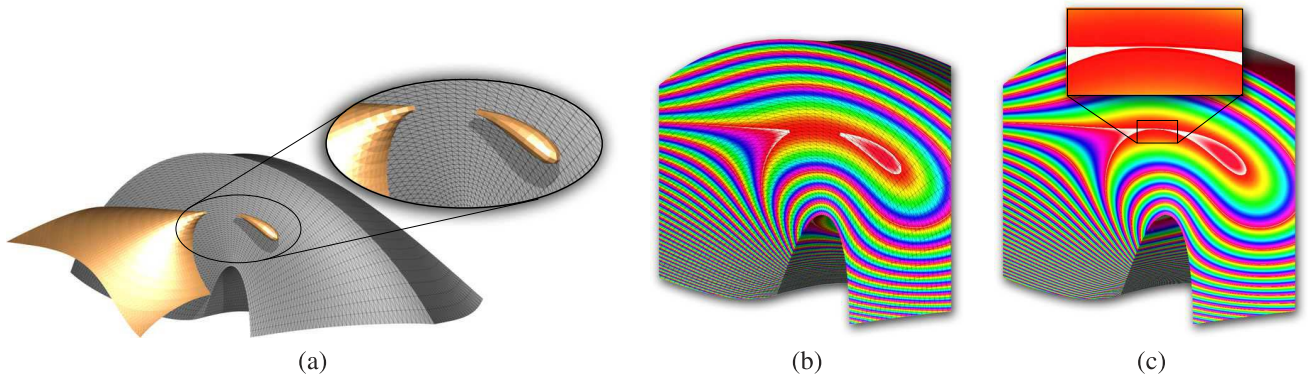

Fig. 2. Discretization of domain from Fig. 1c with 300,000 tetrahedra and application of Marching Tetrahedra (using ParaView). (a) Isosurface. (b) Scalar field on tetrahedra. (c) Our approach on a single triquintic NURBS patch.

and Scheidegger [40] propose an advancing-front method for constructing manifold isosurfaces with well-shaped triangles (Fig. 3), although it has some difficulties when the front meets itself (the stitching problem). Meyer et al. [32] propose a particle system on high-order finite element mesh (arbitrary geometric topology), which applies surface reconstruction on the particles to construct the isosurface mesh; however, the visualization produced is not a watertight surface. When the data are known only at the corners of a hexahedral mesh, our method constructs an approximation by filtering the data with a high-order approximating or interpolating trivariate B-spline filter (see [29]). The filter can be trilinear (only $C^{(0)}$), tricubic ($C^{(2)}$), or higher degree, as required by the user. Then, an isosurface of the high-order approximation is directly rendered with pixel accuracy.

In category 3, the isosurface is rendered directly, i.e., for every pixel on the image plane, its corresponding point on the isosurface is determined (Fig. 3, left). Once the point on the isosurface for a given pixel is known, the pixel can be shaded using the gradient as the normal for the given point. Another motivation to visualize specific isosurfaces is to color-code information, such as material density, to get a better understanding through which materials the isosurface passes. Knoll et al. [24] use a trilinear reconstruction filter on a structured grid and a ray-based octree approach to render isosurfaces and achieve interactive frame rates. Nelson and Kirby [34] propose a ray-based isosurface-rendering algorithm for high-order finite elements using

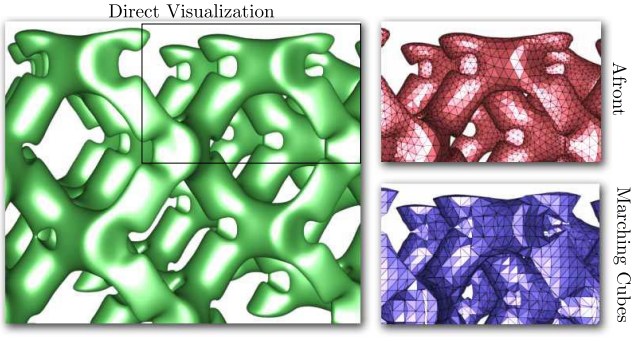

Fig. 3. Isosurface from silicium data set (volvis.org), isovalue of 130 using Marching Cubes (using ParaView), Afront ($\rho = 0.3$), and direct visualization with our proposed method.

classic root-finding methods, but it did not consider element curvature (i.e., the multiple entry and exit problem). Kloetzli et al. [22] construct a set of structured Bézier tetrahedra from a uniform grid to approximate any reconstruction filter with arbitrary footprint. Given this reconstruction, generated from gridded input data (e.g., medical or simulation data), they directly visualize isosurfaces using the ray/isosurface intersection method presented by Loop and Blinn [27].

The method proposed in this paper is most closely related to class 3 approaches, i.e., our proposed method directly visualizes an isosurface from a trivariate NURBS of arbitrary geometric complexity as shown in Fig. 1. However, instead of following only a ray-based scheme, our approach computes the intersection between a ray frustum and the isosurface. Furthermore, it is often desired to visualize the geometry represented by the NURBS. While approaches similar to the work in [1] can be used to render the object-surface geometry, our approach can be used to simultaneously visualize both the geometry represented by the NURBS and the visualization of isosurfaces of the attribute representation (see Fig. 1b) in a robust way. Intersecting a ray frustum with an object in the scene is related to the approaches that propose cone tracing given in the work [2] and beam tracing (see [16]) for more efficient antialiasing, soft shadows, and reflections. However, both of those techniques deal only with polygonal objects. For isosurfaces of algebraic functions, the thesis [10] presents interval approaches to create intersection tests in the ray tracing of implicit surfaces. In particular, it shows a ray sampling-based method to exploit the coherence of rays to accelerate the process of ray tracing implicit surfaces, which can also be used for antialiasing isosurface silhouettes.

2.1 Trivariate NURBS

A trivariate tensor product NURBS mapping is a parametric map $\mathcal{V} : [a_1, a_2] \times [b_1, b_2] \times [c_1, c_2] \rightarrow \Omega \subset \mathbb{R}^3$ of degree $\mathbf{d} = (d_1, d_2, d_3)$ with knot vectors $\tau = (\tau_1, \tau_2, \tau_3)$, defined as

$$\mathcal{V}(\mathbf{u}) := \frac{\sum_{\mathbf{i}=1}^{\mathbf{n}} w_{\mathbf{i}}\, \mathbf{c_i}\, \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})}{\sum_{\mathbf{i}=1}^{\mathbf{n}} w_{\mathbf{i}}\, \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})} \tag{1}$$

$$= \left(\frac{x(\mathbf{u})}{w(\mathbf{u})}, \frac{y(\mathbf{u})}{w(\mathbf{u})}, \frac{z(\mathbf{u})}{w(\mathbf{u})}\right), \tag{2}$$

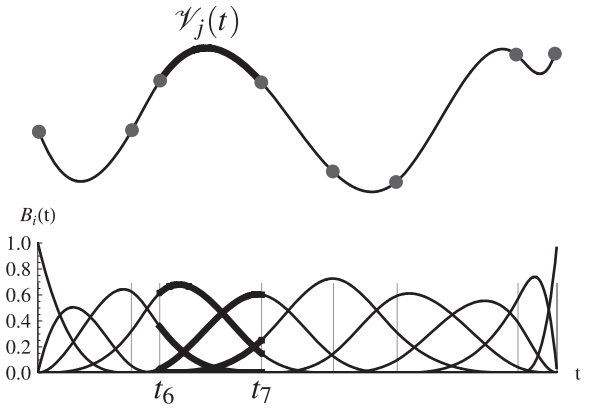

Fig. 4. Cubic NURBS curve with nonuniform knot vector and open end conditions.

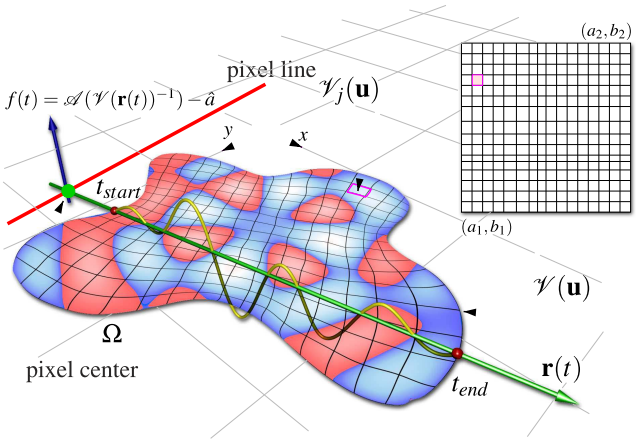

Fig. 5. 2D analogy: ray passing through a bivariate NURBS surface with color-coded attribute field $\mathcal{A}(\mathbf{u})$ intersecting isocontour at roots of $f(t)$, where the red points refer to entry and exit points with the surface.

where $\mathbf{c_i} \in \mathbb{R}^3$ are the control points with associated weights $w_\mathbf{i}$ of the $n_1 \times n_2 \times n_3$ control grid, $\mathbf{i} = (i_1, i_2, i_3)$ is a multi-index, and $\mathbf{u} = (u_1, u_2, u_3)$ is a trivariate parameter value. Every coefficient $\mathbf{c_i}$ has an associated trivariate B-spline basis function $\mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u}) = \prod_{j=1}^{3} B_{i_j,d_j,\tau_j}(u_j)$.

$B_{i_j,d_j,\tau_j}(u_j)$ are linearly independent piecewise polynomials of degree d_j with knot vector $\tau_j = \{t_k^j\}_{k=1}^{n_j+d_j}$. They have local support and are $C^{(d_i-1)}$. Furthermore, $\sum_{\mathbf{i}=1}^{\mathbf{n}} \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u}) = 1$ (see [7]). Fig. 4 illustrates these definitions for the 1D case.

$\mathbf{c_i} \in \mathbb{R}^3$, $\mathcal{V}(\mathbf{u})$ describes the physical geometry and is referred to as the *geometric mapping*. Suppose an attribute $\mathcal{A}(\mathbf{u})$ is related to $\mathcal{V}(\mathbf{u})$ where the attribute function $\mathcal{A} : [a_1, a_2] \times [b_1, b_2] \times [c_1, c_2] \rightarrow \mathbb{R}^{(k)}$ can be formulated as

$$\mathcal{A}(\mathbf{u}) := \frac{\sum_{\mathbf{i}=1}^{\mathbf{n}} w_\mathbf{i}\, a_\mathbf{i}\, \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})}{\sum_{\mathbf{i}=1}^{\mathbf{n}} w_\mathbf{i}\, \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})} \tag{3}$$

$$= \frac{a(\mathbf{u})}{w(\mathbf{u})}, \tag{4}$$

where $\mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})$ is defined as above.

Let $\mathcal{V}_\mathbf{i}(\mathbf{u})$ and $\mathcal{A}_\mathbf{i}(\mathbf{u})$ refer to the geometry and attribute mapping of the $\mathbf{i}$th knot span, $\mathbf{i} = (i_1, i_2, i_3)$, called a "patch," i.e., its parametric domain is $[t^1_{i_1}, t^1_{i_1+1}) \times [t^2_{i_2}, t^2_{i_2+1}) \times [t^3_{i_3}, t^3_{i_3+1})$, where

$$\mathcal{V}_i(\mathbf{u}) := \left(\frac{x_\mathbf{i}(\mathbf{u})}{w_\mathbf{i}(\mathbf{u})}, \frac{y_\mathbf{i}(\mathbf{u})}{w_\mathbf{i}(\mathbf{u})}, \frac{z_\mathbf{i}(\mathbf{u})}{w_\mathbf{i}(\mathbf{u})}\right), \mathcal{A}_i(\mathbf{u}) := \frac{a_\mathbf{i}(\mathbf{u})}{w_\mathbf{i}(\mathbf{u})}. \tag{5}$$

For the purpose of clarity, we consider only scalar attributes, although this approach works equally well for vector attributes. $\mathcal{V}_\mathbf{i}(\mathbf{u})$ and $\mathcal{A}_\mathbf{i}(\mathbf{u})$ are each a single trivariate tensor product polynomial (or rational), and $\mathbb{G} := \{(\mathcal{V}_\mathbf{i}(\mathbf{u}), \mathcal{A}_\mathbf{i}(\mathbf{u}))\}_\mathbf{i}^{\mathbf{n}-\mathbf{d}}$ is the set of geometry and attribute patches, respectively. Note that each geometry patch $\mathcal{V}_\mathbf{i}(\mathbf{u})$ has a corresponding attribute patch $\mathcal{A}_\mathbf{i}(\mathbf{u})$. Furthermore, in case Ω cannot be represented using a single mapping $\mathcal{V}(\mathbf{u})$, then Ω is represented as a collection of the mappings $\mathcal{V}(\mathbf{u})$ and $\mathcal{A}(\mathbf{u})$.

Fig. 5 illustrates these definitions with a single NURBS surface representing $\Omega \in \mathbb{R}^2$.

2.2 Classical Problem Statement

Let $\Omega \in \mathbb{R}^3$ be the domain of interest and $g(x, y, z)$ where $g : \Omega \rightarrow \mathbb{R}$ is an attribute function. In isosurface visualization, the user specifies an isovalue $\hat{a}$ at which to inspect the implicit isosurface of $g(x, y, z) - \hat{a} = 0$. By referring to Fig. 5 (showing the 2D scenario), in ray-based visualization techniques, the ray, passing through the center of a pixel, is represented as $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d}$, where $\mathbf{o}$ is the origin of the ray (location of the eye) in $\mathbb{R}^3$, $\mathbf{d}$ the direction of the ray, and $t \in \mathbb{R}$ the ray parameter. One wants to find the set of t-values that satisfy $f(t) = 0$, where $f(t) = g(\mathbf{r}(t)) - \hat{a}$.

When Ω represents a uniform scalar grid, efficient and interactive methods exist to directly visualize isosurfaces, including a GPU approach to visualize trivariate splines with respect to tetrahedral partitions that transform each patch to its Bernstein-Bézier form [20]. Earlier, a direct rendering paradigm of trivariate B-spline functions for large data sets with interactive rates was presented in the work in [38], where the rendering is conducted from a fixed viewpoint in two phases suitable for sculpting operations. Entezari et al. [14] derive piecewise linear and piecewise cubic box spline reconstruction filters for data sampled on the body-centered cubic lattice. Given such a representation, they directly visualize isosurfaces. Similarly, Kim et al. [21] introduce a box spline approach on the face-centered cubic (FCC) lattice and propose a reconstruction algorithm that can interpolate or approximate the underlying function based on the FCC and directly visualize isosurfaces.

In the case where $g(x, y, z)$ describes an algebraic function in $\mathbb{R}^3$, Blinn [4] uses a hybrid combination of univariate Newton-Raphson iteration and regular falsi. More recently, Reimers and Seland [39] developed an algorithm to visualize algebraic surfaces of high degree, using a polynomial form that yields interactive frame rates on the GPU. Toledo et al. [9] present GPU approaches to visualize algebraic surfaces on the GPU. Interval analysis [33] has been adopted by Hart [15] and recently by Knoll et al. [23] to visualize isosurfaces of algebraic functions as well.

In the following discussion, let $\mathcal{V}(\mathbf{u})$ represent a general domain of interest Ω together with an attribute field $\mathcal{A}(\mathbf{u})$. In this case, Ω is not a cube which has undergone none or at most an affine transformation. Therefore, $g(x, y, z) := \mathcal{A}(\mathcal{V}^{-1}(x, y, z))$. $\mathcal{V}^{-1}(x, y, z)$ is the inverse of a nonidentity and nonaffine mapping, i.e., it cannot be represented in closed form and in order to evaluate the corresponding $f(t)$, the inverse of

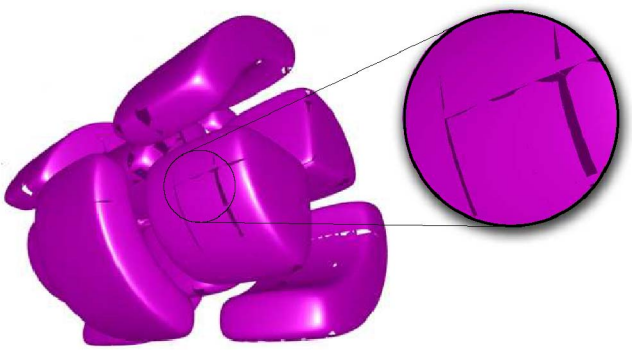

Fig. 6. On the left, piecewise trivariate cubic Bézier patches results in *black pixel* artifacts, due to degenerate derivative at the Bézier patch edges.

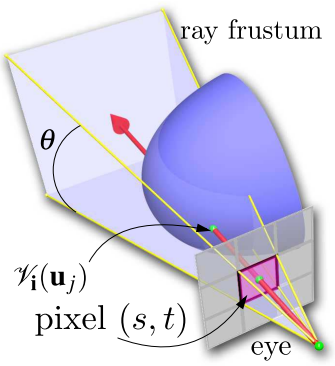

Fig. 7. Ray frustum/isosurface intersection for pixel (s,t) shaded in magenta with adjacent pixels shaded in gray.

$\mathcal{V}^{-1}(x,y,z)$ has to be computed using a root-solving method. Because of this, it is not clear how these methods can be extended to work with the nonlinear, nonpolynomial mapping $\mathcal{V}^{-1}(x,y,z)$. Computing all the roots along $\mathbf{r}(t)$ with those methods would involve reapplication of the respective visualization algorithm, making extensions of such approaches computationally intractable.

Before any root solving takes place, the set $\mathbb{I} \subset \mathbb{G}$ is computed where the geometric subpatches $\mathcal{V}_{\mathbf{i}}(\mathbf{u}) \in \mathbb{I}$ might get intersected by $\mathbf{r}(t)$ and contain the respective isosurface. Finding the roots of $f(t)$ is equivalent to finding the roots of $f_{\mathbf{i}}(t)$ of the geometry patches $\mathcal{V}_{\mathbf{i}}(\mathbf{u}) \in \mathbb{I}$, where

$$f_{\mathbf{i}}(t) := \mathcal{A}_{\mathbf{i}}\big(\mathcal{V}_{\mathbf{i}}^{-1}(\mathbf{r}(t))\big) - \hat{a} = 0. \tag{6}$$

Solving (6) requires finding the range of values of t where $f_{\mathbf{i}}(t)$ is defined, i.e., t-values which correspond to the entry and exit points of $\mathbf{r}(t)$ into $\mathcal{V}_{\mathbf{i}}^{-1}(\mathbf{r}(t))$. Depending on the geometric complexity of Ω, this range can consist of multiple disjoint intervals where each interval is defined by an entry and exit point of the ray with $\mathcal{V}_{\mathbf{i}}(\mathbf{u})$.

One way to compute these intervals is to use the Bézier clipping method proposed in the work [35] on the six sides of the elements in $\mathbb{I}$, implying that the elements in $\mathbb{I}$ have to be turned into Bézier patches using knot insertion (see [7]). While Bézier clipping is an elegant way to visualize Bézier surfaces, it has problems at silhouette pixels. A discussion of its problems and proposed solutions can be found in [11]. Once these pairs of entry and exit points are computed, a numerical root-solving technique, such as the Newton-Raphson method or bisection method, is applied to $f_{\mathbf{i}}(t)$ for each pair. The limitations of these classic methods are well known. That is, Newton's method requires an initial starting value close to the root and depends on $f'_{\mathbf{i}}(t)$, so it fails at degeneracies and where the derivative is close to zero. Krawczyk [25] presents a Newton-Raphson algorithm that uses interval arithmetic for the initial guess. Toth [44] applies this method to render parametric surfaces. However, since Newton's method needs the derivative of $f_{\mathbf{i}}(t)$, it can fail at the edges of $\mathcal{V}_{\mathbf{i}}(\mathbf{u})$ as discussed in [1], leading to the well-known *black pixel* artifacts at the patch boundaries, as shown in Fig. 6. The bisection method is more robust but converges only linearly. The main problem with the bisection method is that the signs of $f_{\mathbf{i}}(t)$ at the entry and exit points must be different, a requirement which often cannot be fulfilled. In summary, an approach which attempts to solve (6) can fail when finding the entry and exit points, finding the inverse $\mathcal{V}_{\mathbf{i}}^{-1}(x,y,z)$, or finding the roots of $f_{\mathbf{i}}(t)$ fails. Furthermore, there is no guarantee of determining all intersections between the isosurface and the area corresponding to the pixel, i.e., it may only determine the intersections at the ray itself.

Another standard approach to intersect a ray $\mathbf{r}(t)$ with an isosurface, as defined in the work by [42], is to solve the system of four equations and four unknowns:

$$\begin{pmatrix} r_x(t) \\ r_y(t) \\ r_z(t) \\ \mathcal{A}(\mathbf{u}) \end{pmatrix} = \begin{pmatrix} x(\mathbf{u}) \\ y(\mathbf{u}) \\ z(\mathbf{u}) \\ \hat{a} \end{pmatrix},$$

where $r_x(t)$, $r_y(t)$, and $r_z(t)$ are the x-, y-, and z-coordinates of $\mathbf{r}(t)$, respectively. Such a nonlinear system can be solved using the general geometric constraint-solving approach proposed by Elber and Kim [13] that uses subdivision and higher dimensional Axis-Aligned Bounding Box (AABB) tests to find a solution where $\mathbf{r}(t)$ and $\mathcal{V}(\mathbf{u})$ are piecewise polynomial or piecewise rational. Elber and Kim applied their approach to bisectors, ray traps, sweep envelopes, and regions accessible during 5-axis machining, but not to rendering isosurfaces. However, as we propose here, pixel-exact isosurface visualization requires further augmentation of the algorithm.

In the following approach, we develop a formulation for a guaranteed determination of all intersections between a ray frustum and an isosurface. The proposed method computes the set of roots simultaneously, avoiding any computation of intervals on which $f_{\mathbf{i}}(t)$ is defined.

3 Mathematical Formulation

In this section, we develop the mathematical formulation that is used to intersect a ray frustum (Fig. 7) with the implicit isosurface $\mathcal{A}(\mathbf{u}) - \tilde{a} = 0$ embedded within $\hat{\mathcal{V}}(\mathbf{u})$, which can represent arbitrary geometry. $\tilde{a}$ is the scalar value for which the isosurface will be visualized.

In the following, we assume the coefficients $\mathbf{c}_{\mathbf{i}}$ and the corresponding weights $w_{\mathbf{i}}$, as defined in Section 2.1, are in eye space, i.e., the camera frustum sits at the origin, pointing down the negative z-axis. Let $\mathbf{P}$ be the 4×4 projection matrix defining the camera frustum, where

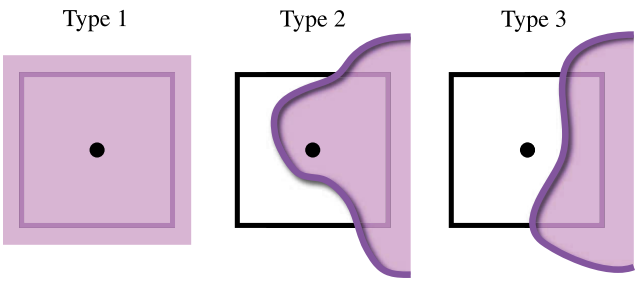

Fig. 8. Three ray frustum/isosurface intersection types: 1) ray frustum and corresponding pixel is fully covered; 2) isosurface silhouette intersects ray frustum with ray intersecting isosurface; 3) Same as Type 2 but ray does not intersect isosurface.

$$\mathbf{P} = \begin{pmatrix} near & 0 & 0 & 0 \\ 0 & near & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2\,far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \quad (7)$$

In this case, $\mathbf{P}$ defines a frustum with a near plane of $near$ units away from the eye with a size of $[-1,1]\times[-1,1]$, and a far plane of far units away from the eye, where $near < far$. Furthermore, $\mathbf{P}$ projects along the z-axis.

$\mathbf{P}$ transforms the frustum and all geometry from eye space into perspective space, i.e., the frustum is transformed into the unit cube $[-1,1]^3$ and every ray frustum in eye space is transformed into a ray box in perspective space. Coefficients $\mathbf{c_i}$ and weights $w_\mathbf{i}$ are transformed into perspective space by

$$(\hat{w}_\mathbf{i}\hat{x}_\mathbf{i}, \hat{w}_\mathbf{i}\hat{y}_\mathbf{i}, \hat{w}_\mathbf{i}\hat{z}_\mathbf{i}, \hat{w}_\mathbf{i})^T = \mathbf{P} \circ (w_\mathbf{i}x_\mathbf{i}, w_\mathbf{i}y_\mathbf{i}, w_\mathbf{i}z_\mathbf{i}, w_\mathbf{i})^T, \quad (8)$$

where $\hat{c}_\mathbf{i} = (\hat{x}_\mathbf{i}, \hat{y}_\mathbf{i}, \hat{z}_\mathbf{i})$ and

$$\begin{pmatrix} \hat{x}_\mathbf{i} \\ \hat{y}_\mathbf{i} \\ \hat{z}_\mathbf{i} \\ \hat{w}_\mathbf{i} \end{pmatrix} = \begin{pmatrix} (near*x_\mathbf{i})/z_\mathbf{i} \\ -(near*y_\mathbf{i})/z_\mathbf{i} \\ \frac{(2*far*near+(far+near)\ z_\mathbf{i})}{(far-near)*z_\mathbf{i}} \\ -w_\mathbf{i}\ z_\mathbf{i} \end{pmatrix}. \quad (9)$$

From that,

$$\hat{\mathcal{V}}(\mathbf{u}) := \frac{\sum_{\mathbf{i}=1}^{\mathbf{n}} \hat{w}_\mathbf{i}\ \hat{\mathbf{c}}_\mathbf{i}\ \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})}{\sum_{\mathbf{i}=1}^{\mathbf{n}} \hat{w}_\mathbf{i}\ \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})} \quad (10)$$

$$= \left(\frac{\hat{x}(\mathbf{u})}{\hat{w}(\mathbf{u})}, \frac{\hat{y}(\mathbf{u})}{\hat{w}(\mathbf{u})}, \frac{\hat{z}(\mathbf{u})}{\hat{w}(\mathbf{u})}\right) \quad (11)$$

is $\mathcal{V}(\mathbf{u})$ in perspective space. Furthermore, let $\hat{\mathbf{x}} = (\hat{x}, \hat{y}, \hat{z})$ be a point in perspective space. Although the transformed ray frustum mapped from eye space to perspective space is a rectangular parallelepiped, we still call it a *ray frustum* to evoke its shape in eye space.

Given a ray frustum constructed from ray $\mathbf{r}(t)$ as shown in Fig. 7, there are three types of intersections between a ray frustum and the isosurface: 1) the isosurface intersects the four planes of the ray frustum and the isosurface's normals point either toward or away from the eye over the whole frustum and $\mathbf{r}(t)$ passes through the isosurface; 2) $\mathbf{r}(t)$ passes through the isosurface but the ray frustum contains an isosurface silhouette; 3) same as case 2 but $\mathbf{r}(t)$ does not pass through the isosurface. Fig. 8 illustrates these three intersection types.

In types 1 and 2, $\mathbf{r}(t)$ intersects the isosurface and can be detected with ray-isosurface intersection. Type 3 requires a different approach. Note that there are cases for which sampling approaches such as pixel subdivision will fail.

First, we present how to detect type 1 and type 2 cases and then discuss how to detect type 3. For an image with resolution $h \times h$ pixels where h is the number of pixels per row and column, we follow the development of Kajiya [19] to detect types 1 and 2 as

$$x - b_s = 0 \ \text{ and } \ y - b_t = 0 \text{ with } b_k = 2(k/h) - 1 + k/(2h), \quad (12)$$

which are two orthogonal planes in perspective space corresponding to pixel at (s,t) whose intersection defines a ray $\mathbf{r}(t)$ aligned with the unit cube.

Given pixel (s,t),

$$\begin{pmatrix} \hat{\alpha}(\mathbf{u}) \\ \hat{\beta}(\mathbf{u}) \\ \hat{\gamma}(\mathbf{u}) \end{pmatrix} := \frac{1}{\hat{w}(\mathbf{u})} \begin{pmatrix} \hat{x}(\mathbf{u}) \\ \hat{y}(\mathbf{u}) \\ a(\mathbf{u}) \end{pmatrix} - \begin{pmatrix} b_s \\ b_t \\ \tilde{a} \end{pmatrix} \quad (13)$$

is rational B-splines. Note, $a(\mathbf{u})$ is defined in (4).

The following constraints must be satisfied for a ray/isosurface intersection:

$$\begin{pmatrix} |\hat{\alpha}(\mathbf{u})| \\ |\hat{\beta}(\mathbf{u})| \\ |\hat{\gamma}(\mathbf{u})| \end{pmatrix} < \begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, \quad (14)$$

i.e., given a solution $\mathbf{u}$, the corresponding $\hat{\mathcal{V}}(\mathbf{u})$ must lie along the ray and on the isosurface within tolerance of $\varepsilon = 1/(2\,h)$. This ensures that a solution lies within a pixel. Multiplying (14) by $\hat{w}(\mathbf{u})$,

$$\begin{pmatrix} |\alpha(\mathbf{u})| \\ |\beta(\mathbf{u})| \\ |\gamma(\mathbf{u})| \end{pmatrix} < \hat{w}(\mathbf{u}) \begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, \quad (15)$$

where $\alpha_\mathbf{i} = \hat{x}_\mathbf{i} - \hat{w}_\mathbf{i}\ b_s$, $\beta_\mathbf{i} = \hat{y}_\mathbf{i} - \hat{w}_\mathbf{i}\ b_t$, $\gamma_\mathbf{i} = a_\mathbf{i} - \hat{w}_\mathbf{i}\ \tilde{a}$, and $(\alpha(\mathbf{u}), \beta(\mathbf{u}), \gamma(\mathbf{u})) := \sum_{\mathbf{i}=1}^{\mathbf{n}} (\alpha_\mathbf{i}, \beta_\mathbf{i}, \gamma_\mathbf{i})\ \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})$.

Equation (15) is not sufficient to detect every isosurface/ray frustum intersection. If an isosurface silhouette lies within the ray frustum but does not get intersected by $\mathbf{r}(t)$ (type 3), then there is no $\mathbf{u}$ that satisfies (15), even though some part of the isosurface (silhouette) lies within the ray frustum. Let

$$\nu(\mathbf{u}) := J_{\hat{\mathbf{x}}}(\mathbf{u}) \cdot \nabla_\mathbf{u}\mathcal{A}(\mathbf{u}) = \nabla_{\hat{\mathbf{x}}}\mathcal{A}(\mathbf{u}) \quad (16)$$

be the gradient in normal direction of the isosurface at $\mathbf{u}$ in perspective space, where $J_{\hat{\mathbf{x}}}(\mathbf{u})$ is the Jacobian at $\mathbf{u}$ in perspective space; then

$$\hat{\delta}(\mathbf{u}) := \nu(\mathbf{u})_{\hat{z}} \quad (17)$$

$$\hat{\eta}(\mathbf{u}) := \left(\left(\frac{\hat{x}(\mathbf{u})}{\hat{w}(\mathbf{u})}, \frac{\hat{y}(\mathbf{u})}{\hat{w}(\mathbf{u})}, 0\right) \times (\nu(\mathbf{u})_x, \nu(\mathbf{u})_y, 0)\right)_{\hat{z}} \quad (18)$$

are rational B-splines, where $\nu(\mathbf{u})_{\hat{z}}$ is the B-spline representing the $\hat{z}$-component of $\nu(\mathbf{u})$.

With ϵ defined as above, a point $\hat{\mathcal{V}}(\mathbf{u})$ on the isosurface silhouette must satisfy

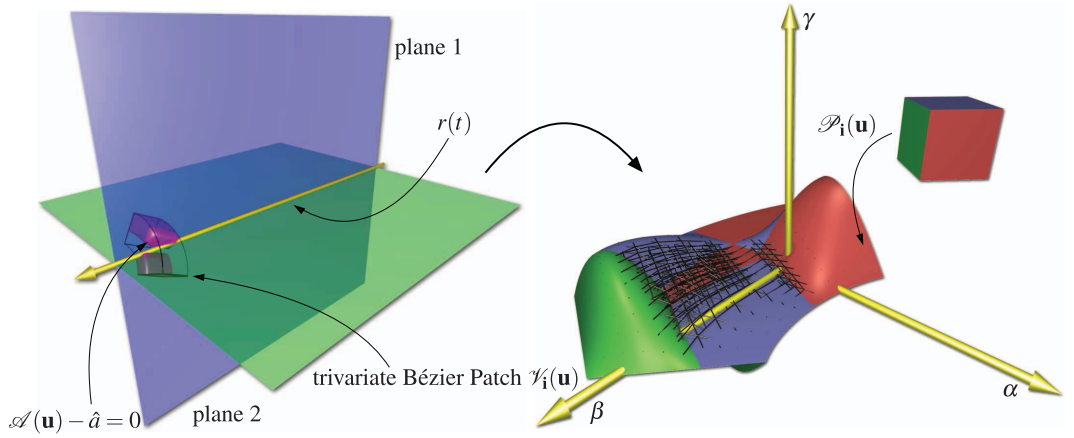

Fig. 9. Left: a ray $\mathbf{r}(t)$, represented as the intersection of two planes, intersects the isosurface $\mathcal{A}(\mathbf{u}) - \hat{a} = 0$ of $\mathcal{V}_\mathbf{i}(\mathbf{u})$. Right: given $\mathcal{V}_\mathbf{i}(\mathbf{u})$, $\mathcal{A}_\mathbf{i}(\mathbf{u})$, and the two planes, a new set of coefficients $Q_\mathbf{k} = (\alpha_\mathbf{k}, \beta_\mathbf{k}, \gamma_\mathbf{k})$ are determined to construct $\mathcal{P}_\mathbf{i}(\mathbf{u})$. The ray intersects the isosurface at $\mathbf{u}_j$ where $|\mathcal{P}_\mathbf{i}(\mathbf{u}_j)|_\infty < \varepsilon$. $\mathcal{P}_\mathbf{i}(\mathbf{u})$ contains self-intersections and degeneracies depending on the number of intersections. The interior of $\mathcal{P}_\mathbf{i}(\mathbf{u})$ is illustrated in wireframe. Parts of the (α, β, γ)-space boundary are formed by the interior of the parametric domain.

$$\begin{pmatrix} |\hat{\delta}(\mathbf{u})| \\ |\hat{\eta}(\mathbf{u})| \\ |\hat{\gamma}(\mathbf{u})| \end{pmatrix} < \begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, \tag{19}$$

i.e., it must lie on the isosurface ($\hat{\gamma}(\mathbf{u}) < \epsilon$), the z-component of the gradient is 0 ($\hat{\delta}(\mathbf{u}) < \varepsilon$), and the isosurface is orthogonal to the ray $\mathbf{r}(t)$ from the center of the pixel ($\hat{\eta}(\mathbf{u}) < \varepsilon$), i.e., the z-component of the cross product between the point and the normal of the isosurface must be zero. Similarly, by multiplying (19) by $\hat{w}(\mathbf{u})$,

$$\begin{pmatrix} |\delta(\mathbf{u})| \\ |\eta(\mathbf{u})| \\ |\gamma(\mathbf{u})| \end{pmatrix} < w(\mathbf{u}) \begin{pmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, \tag{20}$$

where $\delta(\mathbf{u})$ and $\eta(\mathbf{u})$ are defined in terms of the B-spline basis $\mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u})$ and where coefficients $\delta_\mathbf{i}$ and $\eta_\mathbf{i}$ can be computed using Bézier [12] or B-spline [5] multiplication.

Define

$$\mathcal{S}_I := \{\mathbf{u} : (\alpha(\mathbf{u}), \beta(\mathbf{u}), \gamma(\mathbf{u})) = (0, 0, 0)\}. \tag{21}$$

Then, $\mathcal{S}_I$ is the set of $\mathbf{u}$ satisfying (15). $\mathcal{S}_I$ is the set of values where $\mathbf{r}(t)$ intersects the isosurface and is computed such that the set of points $\mathcal{V}(\mathcal{S}_I)$ on the isosurface lie inside the ray frustum corresponding to $\mathbf{r}(t)$ (types 1 and 2). Define $\mathcal{S}_S$ to be the set of $\mathbf{u}$ where $\mathcal{V}(\mathcal{S}_S)$ does not get intersected by $\mathbf{r}(t)$ but a part of an isosurface lies within the ray frustum at $\mathbf{r}(t)$ and that corresponds to a silhouette satisfying the second constraint in (20) (type 3). In the following sections, we present a method to compute the set $\mathcal{S} = \mathcal{S}_I \cup \mathcal{S}_S$.

With this formulation, it is also possible to visualize an isoparametric surface of the geometry mapping $\mathcal{V}(\mathbf{u})$, e.g., $\mathcal{V}(\hat{u}_1, u_2, u_3)$, where $\hat{u}_1$ is fixed, and u_2 and u_3 vary over the parametric domain. This can be achieved by using the NURBS representation to represent fixed parameter values. As an example, in Fig. 2c, $\hat{u}_1 = 0.5$ where u_2 and u_3 vary cutting the respective Ω along u_1 in half. Furthermore, in Fig. 1b, $\hat{u}_3 = 0$ where u_1 and u_2 vary to show only the boundary of Ω representing the Bimba statue.

In the following, we present an efficient subdivision-based solver to compute $\mathcal{S}$.

4 Ray Frustum/Isosurface Intersection

As discussed in Section 3, finding the roots of $f(t)$ is equivalent to determining the set $\mathcal{S}_I$ as defined in (21). To compute all intersections between a ray frustum and the isosurface, the set S_I must be computed. Here, this is achieved through a subdivision approach combined with the Newton-Raphson method.

Before our proposed isosurface intersection is applied, we find the set $\mathbb{I} \in \mathbb{G}$ of candidate geometry subpatches $(\hat{\mathcal{V}}_\mathbf{i}(\mathbf{u}), \hat{\mathcal{A}}_\mathbf{i}(\mathbf{u}))$ that potentially may be intersected by the ray frustum constructed from $\mathbf{r}(t)$ and may contain the isosurface at the isovalue $\tilde{a}$. While the technique itself does not require this step, since the relevant parts can be found through subdivision, we perform it to make the algorithm faster and more efficient. We address different data-dependent ways that $\mathbb{I}$ can be computed in Section 5. In this section, we assume that $\mathbf{r}(t)$ and $\mathbb{I}$ are given. Section 4.1 details our intersection algorithm.

4.1 Algorithm

By following the framework discussed in Section 3, given patch $(\hat{\mathcal{V}}_\mathbf{i}(\mathbf{u}), \hat{\mathcal{A}}_\mathbf{i}(\mathbf{u})) \in \mathbb{I}$ in perspective space, a specified isovalue $\tilde{a}$ and a pixel through whose center the ray $\mathbf{r}(t)$ is passing, the coefficients for the tuple $(\mathcal{P}_\mathbf{i}(\mathbf{u}), \delta_\mathbf{i}(\mathbf{u}))$ are determined, where

$$\mathcal{P}_\mathbf{i}(\mathbf{u}) := \sum_{\mathbf{j}=1}^{\mathbf{d}+1} Q_{\mathbf{j}+\mathbf{i}-1}\, \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u}) = (\alpha_\mathbf{i}(\mathbf{u}), \beta_\mathbf{i}(\mathbf{u}), \gamma_\mathbf{i}(\mathbf{u})), \tag{22}$$

and

$$\delta_\mathbf{i}(\mathbf{u}) := \sum_{\mathbf{j}=1}^{\mathbf{d}+1} \delta_{\mathbf{j}+\mathbf{i}-1}\, \mathcal{B}_{\mathbf{i},\mathbf{d},\tau}(\mathbf{u}), \tag{23}$$

with $Q_{\mathbf{j}+\mathbf{i}-1} = (\alpha_{\mathbf{j}+\mathbf{i}-1}, \beta_{\mathbf{j}+\mathbf{i}-1}, \gamma_{\mathbf{j}+\mathbf{i}-1})$. $\mathcal{P}_\mathbf{i}(\mathbf{u})$ has no direct geometric meaning. We refer the reader to Fig. 9 which shows, on the left side, the two planes defining $\mathbf{r}(t)$, the isosurface, and the boundaries of the tricubic patch. On the right side, it shows the α, β, and γ-coefficients of $\mathcal{P}_\mathbf{i}(\mathbf{u})$ derived from the two planes, the geometry and attribute data. The parametric boundaries transformed by $\mathcal{P}_\mathbf{i}(\mathbf{u})$ are depicted as well, and parts of them may lie in the interior of

the parametric domain of $\mathcal{P}_{\mathbf{i}}(\mathbf{u})$ while forming part of the (α,β,γ)-space boundary.

Given $(\mathcal{P}_{\mathbf{i}}(\mathbf{u}),\delta_{\mathbf{i}}(\mathbf{u}))$, intersecting the ray frustum for ray $\mathbf{r}(t)$ with the isosurface at $\tilde{a}$ is a two-step algorithm.

1. Determine the superset $\mathcal{S}^S=\mathcal{S}^S_I\cup\mathcal{S}^S_S$ of approximate parameter values $\mathbf{u}$, where $\hat{\mathcal{V}}(\mathbf{u})$ lies within the ray frustum and on the isosurface at $\tilde{a}$, using a subdivision procedure with appropriate termination (Section 4.1.1).
2. Apply a filtering process to remove extra parameter values in $\mathcal{S}^S$ that represent the same root (Section 4.2) in order to gain $\mathcal{S}$.

The following discussion details these steps.

4.1.1 Intersection Algorithm

This section presents the core of our ray frustum/isosurface intersection algorithm. Given $(\mathcal{P}_{\mathbf{i}}(\mathbf{u}),\delta_{\mathbf{i}}(\mathbf{u}))$, degeneracies and self-intersections in $\mathcal{P}_{\mathbf{i}}(\mathbf{u})$ at the origin are related to the number of intersections between $\mathbf{r}(t)$ and the isosurface at $\tilde{a}$: assuming there are n intersections, $\mathcal{P}_{\mathbf{i}}(\mathbf{u})$ crosses n times within itself where $\mathcal{P}_{\mathbf{i}}(\mathbf{u})$ evaluates to $(0,0,0)$. Each $\mathbf{u}$ corresponding to an intersection is an element in $\mathcal{S}^S_I$. These cases refer to interactions of types 1 and 2 as illustrated in Fig. 8.

Intersections of type 3 (see Fig. 8) are detected by examining the signs of the coefficients of $\delta_{\mathbf{i}}(\mathbf{u})$. The $\mathbf{u}$s corresponding to these intersections are elements in $\mathcal{S}^S_S$.

The set $\mathcal{S}^S=\mathcal{S}^S_I\cup\mathcal{S}^S_S$ is computed as follows: the fundamental idea of our subdivision procedure is to subdivide $(\mathcal{P}_{\mathbf{i}}(\mathbf{u}),\delta_{\mathbf{i}}(\mathbf{u}))$ in all three directions at the center of its domain, which results in eight subpatches defined by the tuple $(\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}),\delta_{\mathbf{i},\ell,k}(\mathbf{u}))=((\alpha_{\mathbf{i},\ell,k}(\mathbf{u}),\beta_{\mathbf{i},\ell,k}(\mathbf{u}),\gamma_{\mathbf{i},\ell,k}(\mathbf{u})),\delta_{\mathbf{i},\ell,k}(\mathbf{u}))$, where $k=1\ldots8$ identifies the kth subpatch and ℓ refers to the current subdivision level; and

1. adds subpatches $(\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}),\delta_{\mathbf{i},\ell,k}(\mathbf{u}))$ whose enclosing bounding volume contains the origin $\mathbf{0}=(0,0,0)$ to a list $\mathbb{L}$ and
2. examines subpatches $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ whose corresponding isosurface does not get intersected by $\mathbf{r}(t)$, but for which the corresponding isosurface potentially intersects the ray frustum (Section 4.1.2).

Depending on the geometric representation, the algorithm uses either Bézier subdivision or knot insertion [7].

The patches added to $\mathbb{L}$ in Case 1 potentially contain solutions which lie in $\mathcal{S}^S_I$. Patches examined for Case 2 potentially also contain solutions which lie in $\mathcal{S}^S_S$, i.e., Case 3 solutions. Due to properties of B-splines, note that the patch is always contained in the convex hull of its control points, and as the mesh of parametric intervals is split into half, the subdivided control mesh converges quadratically to $\mathcal{P}(\mathbf{u})$.

This procedure is recursively applied to the elements in $\mathbb{L}$ by adding new subdivision patches and removing the corresponding parent patch $(\mathcal{P}_{\mathbf{i},\ell-1,k}(\mathbf{u}),\delta_{\mathbf{i},\ell-1,k}(\mathbf{u}))$. The recursion terminates when all intersections identified with the remaining patches in $\mathbb{L}$ can be determined using the Newton-Raphson method, by using the node location (see [7]) corresponding to the coefficient in $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ closest to $\mathbf{0}$ as an initial starting value. Note that initially $(\mathcal{P}_{\mathbf{i},1,1}(\mathbf{u}),\delta_{\mathbf{i},1,1}(\mathbf{u}))\coloneqq(\mathcal{P}_{\mathbf{i}}(\mathbf{u}),\delta_{\mathbf{i}}(\mathbf{u}))$ and $\mathbb{L}=\{(\mathcal{P}_{\mathbf{i},1,1}(\mathbf{u}),\delta_{\mathbf{i},1,1}(\mathbf{u}))\}$; this strategy is related to the general constraint-solving technique proposed by Elber et al. [13].

Given a subpatch $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$, a crucial issue is whether it contains the origin $\mathbf{0}$ or not. Since $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ can contain self-intersections and geometric complexity in the (α,β,γ)-space, this test is difficult to perform efficiently. The general constraint-solving technique in [13] looks at the signs of the coefficients in $\alpha_{\mathbf{i},\ell,k}(\mathbf{u})$, $\beta_{\mathbf{i},\ell,k}(\mathbf{u})$, and $\gamma_{\mathbf{i},\ell,k}(\mathbf{u})$ independently; that is, it investigates the properties of its AABB in the (α,β,γ)-space. Instead, we examine the geometry of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ in the (α,β,γ)-space more closely. An approximate answer to the $\mathbf{0}$-inclusion test can be given by analyzing the convex hull property of NURBS [7]: if $\mathbf{0}$ does not lie within a convex set, computed from the coefficients $(\alpha_{\mathbf{k}},\beta_{\mathbf{k}},\gamma_{\mathbf{k}})$ defining $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$, then $\mathbf{0}\notin\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$. However, this implies that while $\mathbf{0}$ lies within the convex boundary volume, it may not lie within its corresponding $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$. Thus, during the subdivision process, the number of elements in $\mathbb{L}$, $|\mathbb{L}|$, which contain $\mathbf{0}$, is growing or shrinking. Therefore, $\mathbb{L}$ represents a list of *potential* candidate patches which may contain $\mathbf{0}$. $|\mathbb{L}|$ at a given subdivision level ℓ is strongly dependent on how tightly the convex boundaries enclose its corresponding patches $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})\in\mathbb{L}$. The properties of subdivision guarantee that all potential roots are kept in $\mathbb{L}$.

Generally, it can be said that given $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$'s coefficients $(\alpha_{\mathbf{k}},\beta_{\mathbf{k}},\gamma_{\mathbf{k}})$, a tighter convex boundary volume (e.g., convex hull) is more expensive to compute than a loose convex boundary volume (e.g., AABB), with the cost of our Oriented Bounding Box (OBB) somewhere in the middle. Given a tighter boundary volume, it is generally more expensive to test whether the origin is included in it or not. On the other hand, a tighter convex boundary will have fewer elements in $\mathbb{L}$, resulting in fewer subdivisions. Since a single subdivision step has a running time of $O((d+1)^3)$ where d is the largest degree of the three parametric directions, it is desirable to keep the number of elements in $\mathbb{L}$ as small as possible, especially as d increases. In such a scenario, a good trade-off respecting these opposing aspects is desired. Given the coefficients $(\alpha_{\mathbf{k}},\beta_{\mathbf{k}},\gamma_{\mathbf{k}})$ of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$, while the computation of the convex hull is more expensive compared to much cheaper computation of an AABB, it encloses the coefficients $(\alpha_{\mathbf{k}},\beta_{\mathbf{k}},\gamma_{\mathbf{k}})$ much more tightly.

However, by looking locally at $\mathcal{P}_{\mathbf{i}}(\mathbf{u})$ we can adopt a much tighter bounding volume compared to the AABB, while still not as tight as the convex hull. An OBB, oriented along a given coordinate system with axes $(\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3)$, is determined. Let $\mathbf{u}_c$ be the center of the parametric domain of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$. The Jacobian matrix of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ determines the first-order trivariate Taylor series. We select two of its three directions with the two largest magnitudes to form the main plane of the bounding box. Without loss of generality, suppose they are $\partial\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)/\partial u_1$ and $\partial\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)/\partial u_2$, respectively. We now form a local orthogonal coordinate system at $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ by setting $\mathbf{v}_1$ to the unit vector in the direction $\partial\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)/\partial u_1$, $\mathbf{v}_3$ is the unit vector in the direction of $\partial\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)/\partial u_1\times\partial\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)/\partial u_2$, and $\mathbf{v}_2=\mathbf{v}_3\times\mathbf{v}_1$. As in other applications, the final OBB is constructed by projecting the coefficients $(\alpha_{\mathbf{k}},\beta_{\mathbf{k}},\gamma_{\mathbf{k}})$ onto the planes which are

TABLE 1
Average Image Generation Times Using OBB and AABB, Respectively

data set	degree	# patches	μ pixel (per frame)	OBB μ time PCA (per frame)	OBB μ time ours (per frame)	OBB μ subd. (per frame)	OBB μ / σ list size (overall)	AABB μ time (per frame)	AABB μ subd. (per frame)	AABB μ / σ list size (overall)
Cylinder	tc/tc	$5 \times 2 \times 5$	57 408	0.29	0.15	299 790	1.89/1.03	0.31	667 000	2.70/2.56
Bimba	tc/tc	$27 \times 45 \times 9$	273 024	0.58	0.27	463 281	1.17/0.49	0.81	2 090 467	1.58/5.42
Teardrop	tq/tq	1	56 078	1.97	0.65	371 304	3.54/1.44	1.87	1 007 734	6.14/3.46
VisHuman	tl/tc	$253 \times 253 \times 253$	51 625	1.06	0.40	278 317	1.04/0.24	0.72	587 194	1.12/4.03
Silicium	tl/tc	$95 \times 31 \times 31$	95 425	0.96	0.43	356 862	1.05/0.24	0.74	738 945	1.16/3.29
Torso	tl/tl	2321045	123 084	1.15	0.83	3 502 902	1.09/0.40	1.43	14 913 568	1.36/6.26
CFD	tl/tc	5736	631 342	1.61	0.77	2 016 399	1.88/1.16	1.02	4 124 430	2.70/3.26

The table also shows the timings (in seconds) for each data set when PCA is used instead of our method to compute the OBBs. The degree column presents degrees for the geometry and attribute mapping ($\mathrm{tl} = \mathrm{trilinear}, \mathrm{tc} = \mathrm{tricubic}, \mathrm{tq} = \mathrm{triquintic}$*); μ is the mean; and σ is the standard deviation. The image resolution is* 512×512.

located at the position $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ and have normals $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $-\mathbf{v}_1$, $-\mathbf{v}_2$, $-\mathbf{v}_3$, respectively.

Note that the evaluation of the derivative does not require additional computation, since it is evaluated from the coefficients computed in the subdivision process. Since $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ is a single trivariate polynomial within a patch, expanding around $\mathbf{u}_c$ is justified because the first-order Taylor series becomes a good approximation as the parametric interval decreases in size. This assumes that the determinants of the Jacobians of the neighborhood around $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ are well behaved, i.e., do not change signs. If $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ contains self-intersections and $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ lies on a place in $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ where $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ folds into itself, then the respective determinant at $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ is equal to zero, even though the magnitudes of the partials $\partial\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)/\partial u_k$, $k = 1, 2, 3$, are well behaved due to the smooth representation of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$. However, with increasing subdivision level ℓ, the determinants of Jacobians of the neighborhood of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ do not change signs.

Since $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ undulates through the origin multiple times depending on the number of intersections between the ray and the isosurface, this approximation is not initially useful because the bounding box is computed from the linear approximation of the Taylor series. But as the interval gets smaller, the quality of the approximation increases and the OBB encloses the coefficients of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}_c)$ more tightly (see Fig. 11).

To compare the quality of this OBB, we used PCA on the coefficients of $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ to compute the orientation of a different OBB-bounding box on the data sets discussed in Section 6. Both PCA and the method discussed above result in the same order of subdivisions per pixel with PCA having slightly fewer subdivisions. However, applying PCA was on average about three times slower than our method. Table 1 shows the concrete timings on various data sets.

Also, with this strategy, the number of elements in $\mathbb{L}$ is much smaller compared to the number of elements in $\mathbb{L}$ if AABB had been used. The reader is referred to Fig. 10, which shows the glancing ray scenario with three intersections from Fig. 9 for subdivision level $\ell = 6$. Using AABBs, on a nonsilhouette pixel of the teardrop data set, $\mathbb{L}$ has 67 elements, while by using our OBBs $\mathbb{L}$ has only 7 elements, significantly reducing subdivision effort and memory consumption. More results are given in Section 6.

Termination. The previous paragraphs discussed the subdivision procedure using our OBB scheme. The termination criteria of this procedure are outlined below by answering the question: At which ℓ should the subdivision procedure terminate? A solution $\mathbf{u}_j \in \mathcal{S}_I^S$ must satisfy two requirements:

1. The patch $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ which corresponds to $\mathbf{u}_j$ must represent only one isosurface piece and must not contain folds or self-intersections so that a final application of Newton's method on $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ finds $\mathbf{u}_j$ as a unique solution.
2. $\hat{\mathcal{V}}_{\mathbf{i}}(\mathbf{u})$ has to lie within the frustum defined by the ray $\mathbf{r}(t)$ and the pixel through which $\mathbf{r}(t)$ passes.

As the number ℓ of subdivision levels increases, the geometric complexity of the patches, in $\mathbb{L}$ in terms of tangling and self-intersections, is reduced. Here, we focus

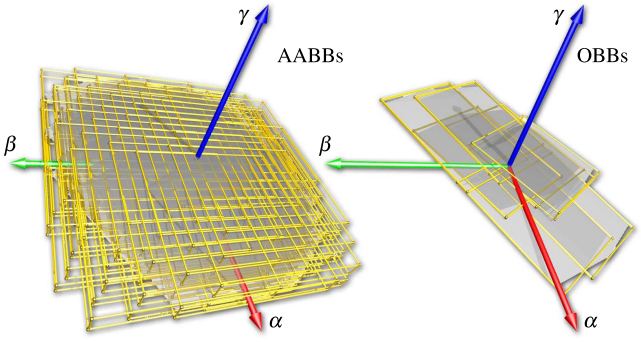

Fig. 10. Subdivision patches stored in $\mathbb{L}$ at subdivision level $\ell = 8$. In this case, the ray glances the isosurface three times, as shown in Fig. 9 involving more extensive subdivision and intersection tests. On the left, AABBs were used which result in $|\mathbb{L}| = 67$. On the right, our OBB computation resulting in $|\mathbb{L}| = 7$, significantly reducing subdivision work.

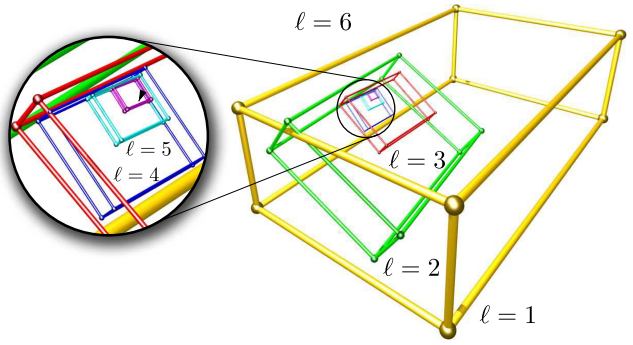

Fig. 11. OBB hierarchy of patches, referring to a ray/isosurface intersection. With growing subdivision level ℓ, the orientation of the OBBs gets closer and closer to its parent's orientation.

on a specific OBB of one $(\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}), \delta_{\mathbf{i},\ell,k}(\mathbf{u})) \in \mathbb{L}$, given a subdivision level ℓ, and examine the signs of the coefficients defining $\delta_{\mathbf{i},\ell,k}(\mathbf{u})$. A sign change means that the isosurface of the patch in perspective space corresponding to $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ potentially faces toward or away from the ray $\mathbf{r}(t)$. This implies that $\mathbf{r}(t)$ intersects the patch at least twice and, therefore, $(\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}), \delta_{\mathbf{i},\ell,k}(\mathbf{u}))$ should be further subdivided. If there is no sign change, then the subdivision process for this patch can be terminated, and Newton's method is used to find the unique solution within the patch, such that

$$\max\Big(\hat{\mathcal{V}}(\mathbf{u}_j) - proj(\hat{\mathcal{V}}(\mathbf{u}_j))\Big) < \varepsilon, \tag{24}$$

where $proj(\hat{\mathcal{V}}(\mathbf{u}_j))$ is the projection of the point $\hat{\mathcal{V}}(\mathbf{u}_j)$ onto $\mathbf{r}(t)$ and $\varepsilon = 1/(2\,h)$ with h as the image resolution (see Section 3). More specifically, given a close enough initial solution $\mathbf{u}_0$, Newton's method tries to iteratively improve the solution and terminates when it is close enough to the exact solution. Close enough in this context means that Newton's method can terminate when the inequality equations, as defined in (15) for a current iterative solution $\mathbf{u}_i$, are satisfied.

In the cases where the initial solution is not good enough for Newton's method, the patch $(\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}), \delta_{\mathbf{i},\ell,k}(\mathbf{u}))$ is further subdivided. This also guarantees that a solution associated with a ray will be within the ray's frustum and does not overlap with adjacent ray frustums. In the rare case that the solution is exactly on the pixel boundary, we use the half-open frustum to guarantee that it is included in only one of the possible adjacent pixels.

4.1.2 Ray Frustum/Isosurface Silhouette Intersection

Before a subpatch $(\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}), \delta_{\mathbf{i},\ell,k}(\mathbf{u}))$ whose OBB does not contain $\mathbf{0}$ is discarded, it must be examined to determine whether the subdomain it covers in $\hat{\mathcal{V}}(\mathbf{u})$ contains any isosurface silhouette intersecting the ray frustum $\mathbf{r}(t)$ in perspective space. If there is no sign change in the coefficients defining either $\gamma_{\mathbf{i},\ell,k}(\mathbf{u})$ or $\delta_{\mathbf{i},\ell,k}(\mathbf{u})$, then the patch can be discarded, because a potential intersection will be caught using the origin-inclusion test (Section 4.1) since in this case the respective isosurface piece completely faces toward or faces away from $\mathbf{r}(t)$.

A sign change in both sets of the coefficients implies that a potential part of the isosurface passes through the ray frustum, facing toward or away from $\mathbf{r}(t)$. If there is such a piece of the isosurface silhouette, then $\mathbf{u}$ is computed so that $\hat{\mathcal{V}}(\mathbf{u})$ lies on the isosurface silhouette and $\mathbf{u}$ is added to $\mathcal{S}_S^S$.

As discussed in Section 3, an isosurface that intersects the frustum (type 3) must have an isosurface silhouette in the frustum, i.e., it must satisfy (20). Given $(\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u}), \delta_{\mathbf{i},\ell,k}(\mathbf{u}))$ with sign changes both in the coefficients defining $\gamma_{\mathbf{i},\ell,k}(\mathbf{u})$ and defining $\delta_{\mathbf{i},\ell,k}(\mathbf{u})$, a patch $\mathcal{Q}_{\mathbf{i},\ell,k}(\mathbf{u})$ is constructed, where

$$\mathcal{Q}_{\mathbf{i},\ell,k}(\mathbf{u}) = (\gamma_{\mathbf{i},\ell,k}(\mathbf{u}), \delta_{\mathbf{i},\ell,k}(\mathbf{u}), \eta_{\mathbf{i},\ell,k}(\mathbf{u})) \tag{25}$$

and the number of self-intersections corresponds to the number of solutions $\mathbf{u}$.

Termination. Subdivision is used to solve $\mathcal{Q}_{\mathbf{i},\ell,k}(\mathbf{u}) = \mathbf{0}$, where the 3D version of the normal cone (NC) test proposed in the work [41] is used to make a faithful decision to stop the subdivision process of patch $\mathcal{Q}_{\mathbf{i},\ell,k}(\mathbf{u})$. This test computes the NCs for the mappings $\gamma_{\mathbf{i},\ell,k}(\mathbf{u})$, $\delta_{\mathbf{i},\ell,k}(\mathbf{u})$, and $\eta_{\mathbf{i},\ell,k}(\mathbf{u})$. Elber et al. show that when the NCs of these three mappings do not intersect, the patch can contain at most one zero. If the NC test fails, i.e., $\mathcal{Q}_{\mathbf{i},\ell,k}(\mathbf{u})$ contains self-intersections, then $\mathcal{Q}_{\mathbf{i},\ell,k}(\mathbf{u})$ is further subdivided. If the NC succeeds, this implies that a subdivided patch does not contain self-intersections. Newton's method is used as above to find a solution $\mathbf{u}$ which is added to $\mathcal{S}_S^S$ when (20) is satisfied.

Note that this additional solution step to find points on an isosurface silhouette within a ray frustum is executed only at isosurface silhouettes, when there are sign changes in the coefficients defining $\gamma_{\mathbf{i},\ell,k}(\mathbf{u})$ and $\delta_{\mathbf{i},\ell,k}(\mathbf{u})$. In most cases, as observed in our experiments, the ray $\mathbf{r}(t)$ intersects the isosurface.

4.2 Filtering Intersection Result

The subdivision procedure discussed in the previous section, applied to the patch $(\mathcal{V}_{\mathbf{i}}(\mathbf{u}), \mathcal{A}_{\mathbf{i}}(\mathbf{u})) \in \mathbb{I}$, outputs the superset $\mathcal{S}^S$ of approximate parameter values $\mathbf{u}_j$, i.e., where $|\mathcal{A}(\mathbf{u}_i) - \hat{a}| < \varepsilon$. By following the framework from Section 3, our method is guaranteed to compute all roots. However, due to the approximate $\mathbf{0}$-inclusion test and the fact that it is a numerical method, it can be the case that $\mathcal{S}^S$ contains multiple solutions that represent the same root. This is because of the use of OBB to determine whether $\mathbf{0}$ is contained in its respective patch. As discussed above, $\mathcal{P}_{\mathbf{i},\ell,k}(\mathbf{u})$ may not contain $\mathbf{0}$ while its OBB contains it. A final postprocess on $\mathcal{S}^S$, yielding the set $\mathcal{S}$, is therefore required for the removal of duplicate solutions.

In the scenario of direct isosurface visualization, multiple cases can appear (shown in Fig. 13, computed solutions in green). In Case I, it can happen that parts of the isosurface lie very close together. Therefore, the corresponding solutions are numerically very similar, even though they represent different solutions. In Case II, the ray might glance or touch the isosurface tangentially, which corresponds to two solutions. In Case III, the usual case, two solutions can represent the same true solution even though they are numerically different. We remove duplicates by examining the derivative of the function $f(t)$ given by

$$f'(t) = \left\langle \frac{\partial \mathbf{r}(t)}{\partial t}, J^{-1} \circ \nabla \mathcal{A}(\mathcal{V}^{-1}(\mathbf{r}(t))) \right\rangle, \tag{26}$$

where J^{-1} is the Jacobian of $\mathcal{V}^{-1}(\mathbf{r}(t))$, and $\circ$ is the matrix/vector product. As the ray $\mathbf{r}(t)$ travels through the volume, it enters and eventually exits the isosurface. Entering means that $\mathbf{r}(t)$ intersects the isosurface at the positive side; this corresponds to a positive derivative of (26) at the corresponding entry location. The exit point refers to a negative derivative of (26). With this observation, Case I can be identified. Case II appears at the silhouette of the isosurface. If $f'(t) \approx 0$, then one of the corresponding solutions can be discarded. For Case III, since the signs of $f'(t)$ for the corresponding solutions are both positive and negative, respectively, one of them can be discarded.

In our implementation, for every $\mathbf{u}_i \in \mathcal{S}^S$, we determine its corresponding t_i by solving the linear equation $t_i = \mathbf{r}^{-1}(\mathcal{V}(\mathbf{u}_i))$ and evaluate $f'(t_i)$. The resulting list of t-values is sorted in increasing order. Finally, the sorted list which corresponds to the order in which the ray travels through

the volume is traversed by removing those elements which violate the rule of alternation of the signs of $f'(t_i)$ within the list. Note that in some rare subpixel cases, incorrect ordering can occur and cause incorrect transparency results. This is a subpixel problem and can be resolved by further subdividing the pixel. However, we found that no visual artifacts result.

This algorithm detects intersections in the pathological case that a whole interval of $\mathbf{r}(t)$ lies on the isosurface. However, as with all numerical methods, there are not ways to determine this analytical condition, but instead find many discrete values of t. We set a heuristic threshold on the maximum number of ray-isosurface intersections per ϵ-length of t. If the number of intersections exceeds it, we use only the smallest value and the largest value.

5 Determining the Set of Intersection Patches

As discussed above, $\mathbb{I} \subset \mathbb{G}$ is the set which contains the geometric subpatches $(\mathcal{V}_\mathbf{i}(\mathbf{u}), \mathcal{A}_\mathbf{i}(\mathbf{u}))$ that intersect the ray frustum constructed from $\mathbf{r}(t)$ and through which the isosurface $\mathcal{A}(\mathbf{u}) - \hat{a} = 0$ passes. There are multiple ways to determine $\mathbb{I}$, which depend on the number of coefficients defining $\mathcal{V}(\mathbf{u})$ and the geometry it describes in physical space. In our implementation, we distinguish between three different types of geometry: 1) general geometry describing a physical domain with a large number of coefficients; 2) general geometry describing a physical domain of interest with few coefficients; and 3) a uniform grid, where $\mathcal{V}_\mathbf{i}(\mathbf{u})$ describes the identity mapping, i.e., $\mathcal{V}_\mathbf{i}(\mathbf{u}) = \mathbf{u}$.

For geometries 1 and 2, we employ a kd-tree as an acceleration structure, where an AABB is computed from the coefficients of $\mathcal{V}_\mathbf{i}(\mathbf{u})$ where $(\mathcal{V}_\mathbf{i}(\mathbf{u}), \mathcal{A}_\mathbf{i}(\mathbf{u})) \in \mathbb{G}$. $\mathbb{I}$ is determined by kd-tree traversal using the traversal algorithm proposed by Sung and Shirley [43], where the ray $\mathbf{r}(t)$ is intersected with the bounding boxes. Note the resulting $\mathbb{I}$ can contain patches that are not intersected by $\mathbf{r}(t)$. If $|\mathbb{G}|$ is small, then the AABBs do not tightly bound $\mathcal{V}_\mathbf{i}(\mathbf{u})$, and $\mathbb{I}$ contains a larger number of patches that do not intersect $\mathbf{r}(t)$. In that case, we apply knot insertion to the elements in $\mathbb{G}$ to turn them into Bézier patches whose corresponding AABBs are much tighter. When $\mathcal{V}(\mathbf{u})$ consists of a large number of coefficients, the ratio between the AABB and its corresponding $\mathcal{V}_\mathbf{i}(\mathbf{u})$ is close to 1. In that case, Bézier conversion is not a significant advantage, but a disadvantage because of its higher memory consumption and preprocessing time. In (3), where $\mathcal{V}(\mathbf{u})$ represents a uniform grid, i.e., when $\mathcal{V}(\mathbf{u}) = \mathbf{u}$, conventional uniform grid traversal is used without any data preprocessing. Also note that in this case (e.g., Fig. 1d), the smooth representation for $\mathcal{A}(\mathbf{u})$ is generated using a B-spline [29] filter to which our method is applied.

6 Analysis and Results

This section is concerned with the correctness and efficiency of our approach. Verifying the correctness of an isosurface visualization technique on acquired data is difficult, especially in terms of correctness of the topology and existence of all features, since given data usually only approximate the true solution (e.g., the results of Galerkin's method or data from a CT scan). In this section, we use the fact that every rational polynomial can be represented with a NURBS representation, i.e., there are coefficients $a_\mathbf{i} \in \mathbb{R}$ such that

$$a(x, y, z) \equiv \mathcal{A}(x, y, z) = \sum_{\mathbf{i}=1}^{\mathbf{n}} a_\mathbf{i}\, \mathcal{R}_{\mathbf{i},\mathbf{d},\tau}(x, y, z), \qquad (27)$$

defined over a rectangular parallelepiped of $\Omega \in \mathbb{R}^3$, where Ω is rectangular and where $a(x, y, z)$ is an algebraic function. Given $a(x, y, z)$ and a NURBS basis (as defined in Section 2.1) whose degree matches the highest degree of $a(x, y, z)$, the coefficients $a_\mathbf{i}$ can be derived by solving the multivariate version of Marsden's identity [30]. If $a(x, y, z)$ is a cubic algebraic function, the approach of Bajaj et al. [3] can be used to compute coefficients $a_\mathbf{i}$ for the NURBS basis. For our tests, we chose the isosurface at 0.0 of the teardrop function, defined as $a(x, y, z) = x^5/2 + x^4/2 - y^2 - z^2$, a common function to test correctness of a visualization technique. The thin features around the origin, as seen in Fig. 1c, are challenging to isosurface meshing techniques where areas around the thin feature are missing (e.g., see work by [36]). Next to the coefficients $a_\mathbf{i}$, our method requires a choice of coefficients $P_\mathbf{i} = (x_\mathbf{i}, y_\mathbf{i}, z_\mathbf{i})$ to define $\mathcal{V}(\mathbf{u})$. If $P_\mathbf{i}$ are node locations as defined in [7], then $a(x, y, z) \equiv \mathcal{A}(x, y, z)$ is achieved. However, since our technique is independent of the geometric complexity, a choice can be made on the mapping $\mathcal{V}(\mathbf{u})$. A more general version of (27) is $a(\mathcal{V}^{-1}(\mathbf{u})) \equiv \mathcal{A}(\mathbf{u})$, in which $a(x, y, z)$ undergoes a nonlinear transformation defined by $\mathcal{V}(\mathbf{u})$ deforming Ω. By referring to Fig. 1c, Ω is stretched and perturbed, which results in a deformation of $a(x, y, z) = 0$. The deformation does not affect the accuracy of our algorithm in reproducing the thin feature discussed above, indicating robustness and topological correctness of our technique at the per-pixel level.

In Fig. 14, the number of subdivisions per pixel of the isosurface intersection technique, using AABBs and OBBs constructed in the above section, is visualized. The images are generated from the same view as the shaded version in Fig. 1. It can be seen that major work is done only for pixels that actually correspond to a point on the isosurface and pixels on the silhouette. When employing an AABB, a large number of silhouette pixels require an average of 270 and up to 380 subdivisions per pixel. With OBBs, only a few pixels require more than 68 subdivisions, and, on average, 35 subdivisions are needed for the silhouette. This means that the number of subdivision levels for OBB is much smaller than with AABB, resulting in a more memory efficient algorithm.

6.1 Timings

Fig. 12a shows the result of our algorithm, rendering geometry of a torso with multiple isosurfaces of the potential trilinear (cubic) field. Both are represented using unstructured hex meshes. In Fig. 12b, we present the visualization of an isosurface of pressure ($\mathrm{isovalue} = 0$) generated due to a rotating canister traveling through an incompressible fluid. The data set was generated by the spectral/hp high-order finite element CFD simulation code,

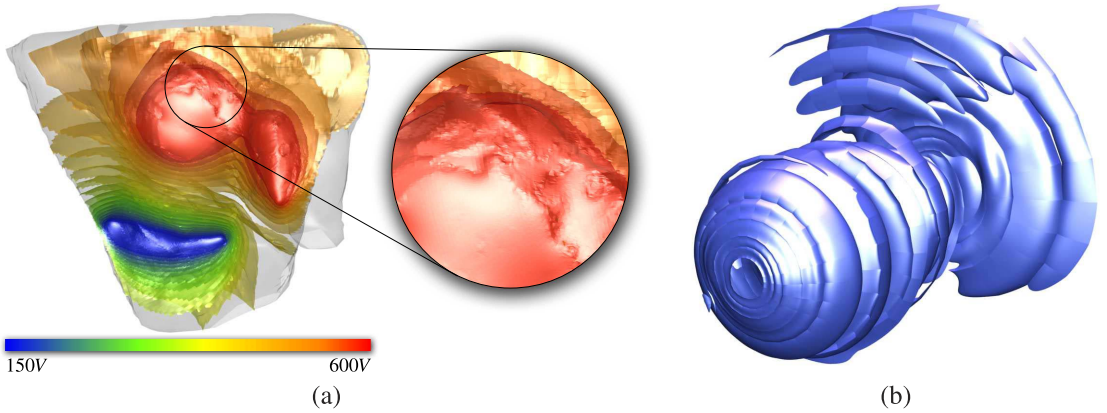

Fig. 12. (a) Unstructured hexahedral mesh (≈ 2.3 million elements) of a segmented torso. Isosurfaces representing voltages of the potential field (using a trilinear basis) are used to specify locations of electrodes to determine efficacy of defibrillation to find a good location to implant a defibrillator into a child. (b) Wake of a rotating canister traveling through a fluid (isosurface of pressure from spectral/hp element CFD simulation data as used in the work [34], [32]). The $C^{(0)}$ nature of the boundaries of the spectral/hp elements can be seen on the isosurface and is not an artifact of our proposed method.

Nektar, and was used as test data set for visualization in the works [32], [34]. The geometry of these data is trilinear ($C^{(0)}$), and the attribute data are tricubic.

Table 1 provides concrete numbers of the proposed approach in comparison to the AABB and PCA as discussed in Section 4. The table provides average render times (μ time), additional information such as the average number of pixels per frame (μ pixel), the average number of subdivisions per frame (μ subd.), the average list size of L overall (μ list size), and the standard deviation of the list size L overall (σ list size). Due to space constraints for PCA, only the render times are presented, since the remaining values are within $\pm 1\%$ compared to our method.

The data in the table were generated by rotating the camera around the respective isosurfaces in 360 frames, using Phong shading and normals computed from the NURBS representation. The above information is generated using our method's OBBs and AABBs from the same space. Subdivision is the major work in both cases. However, both cases outperform the typical problem formulation with the four equations and four unknowns discussed in Section 2, since subdivision has to be performed on four parametric directions with each subdivision being $O((d+1)^4)$ versus three parametric subdivisions with $O((d+1)^3)$ for each subdivision, where d is the degree.

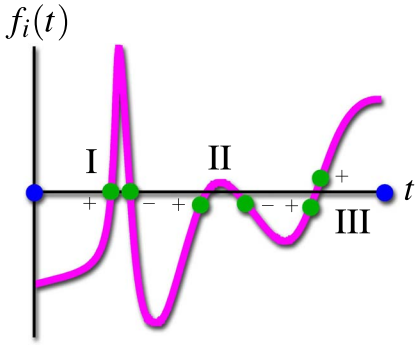

Fig. 13. $\mathcal{S}$ can contain duplicate solutions which can arise due to the scenarios I, II, and III. The derivative of the scalar function $f(t)$ is used to filter $\mathcal{S}$ to identify unique solutions and solutions representing the same root.

The timings were taken on interlinked Intel Xeon X7350 Processors comprised of 32 cores using gcc version 4.3 and OpenMP. Evidently, OBB is up to three times faster than AABB, depending on the isosurface complexity.

7 Conclusion

In this paper, we proposed a novel direct isosurface visualization technique which computes all the intersections between a ray and an isosurface embedded in various representations, such as data-fitted geometry, rational geometry, and uniform grids. Our framework supports rendering the isosurface with view-independent

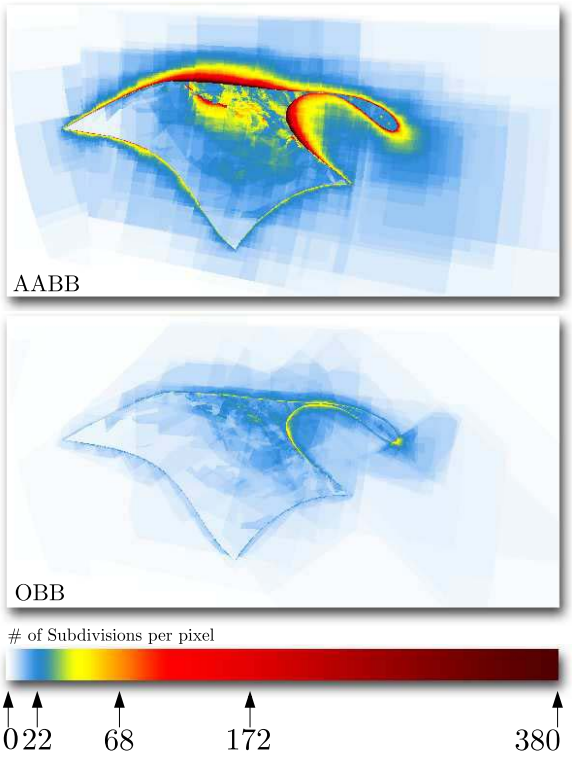

Fig. 14. Number of subdivisions per pixel frustum using AABB and OBB for teardrop isosurface from Fig. 1.

transparency. The technique is robust, user friendly, and easy to implement: all the images in this paper, which show different isosurface visualization scenarios, did not require tweaking and had no parameter readjustment. We have shown that even though the high-order geometry mapping contains parametric distortions (e.g., Fig. 1c), important features in the isosurface are still maintained, something that is challenging for most isosurface techniques. Currently, we are working on a GPU implementation where we expect a significant speed-up of the technique. A direction for future work is to extend the approach to tessellated isosurfaces.

ACKNOWLEDGMENTS

This work was supported in part by ARO W911NF0810517 (Program Manager: Dr. Mike Coyle) and US National Science Foundation (NSF) IIS-1117997. The authors gratefully acknowledge the computational support and resources provided by the Scientific Computing and Imaging Institute at the University of Utah. Data Courtesy of the Torso model is Jeroen Stintra from the Scientific Computing and Imaging Institute at the University of Utah. They would also like to thank Mathias Schott for helpful discussions.

REFERENCES

[1] O. Abert, M. Geimer, and S. Müller, "Direct and Fast Ray Tracing of NURBS Surfaces," *Proc. IEEE Symp. Interactive Ray Tracing,* pp. 161-168, 2006.

[2] J. Amanatides, "Ray Tracing with Cones," *SIGGRAPH Computer Graphics,* vol. 18, no. 3, pp. 129-135, 1984.

[3] C.L. Bajaj, R.L. Holt, and A.N. Netravali, "Rational Parametrizations of Nonsingular Real Cubic Surfaces," *ACM Trans. Graphics,* vol. 17, no. 1, pp. 1-31, 1998.

[4] J.F. Blinn, "A Generalization of Algebraic Surface Drawing," *ACM Trans. Graphics,* vol. 1, no. 3, pp. 235-256, 1982.

[5] X. Chen, R.F. Riesenfeld, and E. Cohen, "Sliding Windows Algorithm for B-Spline Multiplication," *SPM '07: Proc. ACM Symp. Solid and Physical Modeling,* pp. 265-276, 2007.

[6] P. Cignoni, L.D. Floriani, C. Montani, E. Puppo, and R. Scopigno, "Multiresolution Modeling and Visualization of Volume Data Based on Simplicial Complexes," *VVS '94: Proc. Symp. Volume Visualization,* pp. 19-26, 1994.

[7] E. Cohen, R.F. Riesenfeld, and G. Elber, *Geometric Modeling with Splines: An Introduction.* A.K. Peters, Ltd., 2001.

[8] J.A. Cottrell, A. Reali, Y. Bazilevs, and T.R. Hughes, "Isogeometric Analysis of Structural Vibrations," *Computer Methods in Applied Mechanics and Eng.,* vol. 195, nos. 41-43, pp. 5257-5296, 2006.

[9] R. de Toledo, B. Levy, and J.-C. Paul, "Iterative Methods for Visualization of Implicit Surfaces on GPU," *ISVC '07: Proc. Int'l Symp. Visual Computing,* pp. 598-609, Nov. 2007.

[10] J.E.F. Díaz, "Improvements in the Ray Tracing of Implicit Surfaces Based on Interval Arithmetic," PhD thesis, Universitat de Girona, 2008.

[11] A. Efremov, V. Havran, and H.-P. Seidel, "Robust and Numerically Stable Bézier Clipping Method for Ray Tracing NURBS Surfaces," *SCCG '05: Proc. 21st Spring Conf. Computer Graphics,* pp. 127-135, 2005.

[12] G. Elber, "Free Form Surface Analysis Using a Hybrid of Symbolic and Numeric Computation," PhD thesis, Computer Science Departmente, Univ. of Utah, 1992.

[13] G. Elber and M.-S. Kim, "Geometric Constraint Solver Using Multivariate Rational Spline Functions," *SMA '01: Proc. Sixth ACM Symp. Solid Modeling and Applications,* pp. 1-10, 2001.

[14] A. Entezari, R. Dyer, and T. Moller, "Linear and Cubic Box Splines for the Body Centered Cubic Lattice," *VIS '04: Proc. Conf. Visualization,* pp. 11-18, 2004.

[15] J.C. Hart, "Ray Tracing Implicit Surfaces," *Proc. SIGGRAPH Course Notes: Design, Visualization and Animation of Implicit Surfaces,* pp. 1-16, 1993.

[16] P.S. Heckbert and P. Hanrahan, "Beam Tracing Polygonal Objects," *SIGGRAPH '84: Proc. 11th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 119-127, 1984.

[17] J. Hua, Y. He, and H. Qin, "Multiresolution Heterogeneous Solid Modeling and Visualization Using Trivariate Simplex Splines," *SM '04: Proc. Ninth ACM Symp. Solid Modeling and Applications,* pp. 47-58, 2004.

[18] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs, "Isogeometric Analysis: CAD, Finite Elements, NURBS, Exact Geometry, and Mesh Refinement," *Computer Methods in Applied Mechanics and Eng.,* vol. 194, pp. 4135-4195, 2005.

[19] J.T. Kajiya, "Ray Tracing Parametric Patches," *SIGGRAPH '82: Proc. Ninth Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 245-254, 1982.

[20] T. Kalbe and F. Zeilfelder, "Hardware-Accelerated, High-Quality Rendering Based on Trivariate Splines Approximating Volume Data," *Computer Graphics Forum,* vol. 27, no. 2, pp. 331-340, 2008.

[21] M. Kim, A. Entezari, and J. Peters, "Box Spline Reconstruction on the Face-Centered Cubic Lattice," *IEEE Trans. Visualization and Computer Graphics,* vol. 14, no. 6, pp. 1523-1530, Nov./Dec. 2008.

[22] J. Kloetzli, M. Olano, and P. Rheingans, "Interactive Volume Isosurface Rendering Using BT Volumes," *I3D '08: Proc. Symp. Interactive 3D Graphics and Games,* pp. 45-52, 2008.

[23] A. Knoll, Y. Hijazi, C.D. Hansen, I. Wald, and H. Hagen, "Interactive Ray Tracing of Arbitrary Implicit Functions," *Proc. Eurographics/IEEE Symp. Interactive Ray Tracing,* 2007.

[24] A. Knoll, I. Wald, S. Parker, and C. Hansen, "Interactive Isosurface Ray Tracing of Large Octree Volumes," *Proc. IEEE Symp. Interactive Ray Tracing,* pp. 115-124, Sept. 2006.

[25] R. Krawczyk, "Newton Algorithmen Zur Bestimmung Von Nullstellen Mit Fehlerschranken," *Computing,* vol. 4, pp. 187-201, 1969.

[26] M. Levoy, "Efficient Ray Tracing of Volume Data," *ACM Trans. Graphics,* vol. 9, no. 3, pp. 245-261, 1990.

[27] C. Loop and J. Blinn, "Real-Time GPU Rendering of Piecewise Algebraic Surfaces," *ACM Trans. Graphics,* vol. 25, no. 3, pp. 664-670, 2006.

[28] W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *SIGGRAPH Computer Graphics,* vol. 21, no. 4, pp. 163-169, 1987.

[29] S.R. Marschner and R.J. Lobb, "An Evaluation of Reconstruction Filters for Volume Rendering," *VIS '94: Proc. Conf. Visualization,* pp. 100-107, 1994.

[30] M.J. Marsden, "An Identity for Spline Functions with Applications to Variation Diminishing Spline Approximation," *J. Approximation Theory,* vol. 3, pp. 7-49, 1970.

[31] W. Martin and E. Cohen, "Representation and Extraction of Volumetric Attributes Using Trivariate Splines," *Proc. Symp. Solid and Physical Modeling,* pp. 234-240, 2001.

[32] M. Meyer, B. Nelson, R. Kirby, and R. Whitaker, "Particle Systems for Efficient and Accurate High-Order Finite Element Visualization," *IEEE Trans. Visualization and Computer Graphics,* vol. 13, no. 5, pp. 1015-1026, Sept./Oct. 2007.

[33] R.E. Moore, *Interval Analysis.* Prentice-Hall, 1966.

[34] B. Nelson and R.M. Kirby, "Ray-Tracing Polymorphic Multidomain Spectral/hp Elements for Isosurface Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 12, no. 1, pp. 114-125, Jan./Feb. 2006.

[35] T. Nishita, T.W. Sederberg, and M. Kakimoto, "Ray Tracing Trimmed Rational Surface Patches," *SIGGRAPH Computer Graphics,* vol. 24, no. 4, pp. 337-345, 1990.

[36] A. Paiva, H. Lopes, T. Lewiner, and L.H. de Figueiredo, "Robust Adaptive Meshes for Implicit Surfaces," *Proc. Brazilian Symp. Computer Graphics and Image Processing,* pp. 205-212, 2006.

[37] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan, "Interactive Ray Tracing for Isosurface Rendering," *VIS '98: Proc. Conf. Visualization,* pp. 233-238, 1998.

[38] A. Raviv and G. Elber, "Interactive Direct Rendering of Trivariate B-Spline Scalar Functions," *IEEE Trans. Visualization and Computer Graphics,* vol. 7, no. 2, pp. 109-119, Apr.-June 2001.

[39] M. Reimers and J. Seland, "Ray Casting Algebraic Surfaces Using the Frustum Form," *Computer Graphics Forum,* vol. 27, no. 2, pp. 361-370, 2008.

[40] J. Schreiner and C. Scheidegger, "High-Quality Extraction of Isosurfaces from Regular and Irregular Grids," *IEEE Trans. Visualization and Computer Graphics,* vol. 12, no. 5, pp. 1205-1212, Sept./Oct. 2006.
[41] T. Sederberg and A. Zundel, "Pyramids that Bound Surface Patches," *Graphical Models and Image Processing,* vol. 58, no. 1, pp. 75-81, Jan. 1996.
[42] P. Shirley, *Fundamentals of Computer Graphics.* A.K. Peters, Ltd., 2002.
[43] K. Sung and P. Shirley, "Ray Tracing with the BSP Tree," *Graphics Gems III,* pp. 271-274, Academic Press Professional, Inc., 1992.
[44] D.L. Toth, "On Ray Tracing Parametric Surfaces," *SIGGRAPH '85: Proc. 12th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 171-179, 1985.
[45] O. Wilson, A. VanGelder, and J. Wilhelms, "Direct Volume Rendering via 3D Textures," technical report, Univ. of California at Santa Cruz, 1994.
[46] Y. Zhang, Y. Bazilevs, S. Goswami, C.L. Bajaj, and T.J.R. Hughes, "Patient-Specific Vascular NURBS Modeling for Isogeometric Analysis of Blood Flow," *Computer Methods in Applied Mechanics and Eng.,* vol. 196, nos. 29/30, pp. 2943-2959, 2007.

Tobias Martin received the undergraduate degree in computer science (Diplom-Informatiker FH) in 2004 from the University of Applied Sciences, Furtwangen, Germany. He is currently working toward the PhD degree in computer science from the University of Utah, Salt Lake City. His research interests include topics in computer graphics such as geometric modeling, rendering, and visualization.

Elaine Cohen received the BS(*cum laude*) degree in mathematics in 1968 from Vassar College. She received the MS degree in 1970 and the PhD degree in mathematics in 1974 from Syracuse University. She is a professor in the School of Computing, University of Utah, and has been coheading the Geometric Design and Computation Research Group since 1980. She has focused her research on geometric computations for computer graphics, geometric modeling, and manufacturing, with emphasis on complex sculptured models represented using Non-Uniform Rational B-splines (NURBS) and NURBS features.

Robert M. Kirby (M'04) received the MS degree in applied mathematics, the MS degree in computer science, and the PhD degree in applied mathematics from Brown University, Providence, Rhode Island, in 1999, 2001, and 2002, respectively. He is currently an associate professor of computer science with the School of Computing, University of Utah, Salt Lake City, where he is also an adjunct associate professor in the Departments of Bioengineering and Mathematics and a member of the Scientific Computing and Imaging Institute. His current research interests include scientific computing and visualization. He is a member of the IEEE.