Interactive Isocontouring of High-Order Surfaces

Christian Pagot[1], Joachim Vollrath[2], Filip Sadlo[2], Daniel Weiskopf[2], Thomas Ertl[2], and João L. D. Comba[1]

1 **Instituto de Informática, UFRGS**
Porto Alegre, RS, Brazil, CP 15064
{capagot,comba}@inf.ufrgs.br

2 **VISUS, Universität Stuttgart**
Allmandring 19, 70569, Stuttgart, Germany
{vollrath,sadlo,weiskopf,ertl}@visus.uni-stuttgart.de

Abstract

Scientists and engineers are making increasingly use of *hp*-adaptive discretization methods to compute simulations. While techniques for isocontouring the high-order data generated by these methods have started to appear, they typically do not facilitate interactive data exploration. This work presents a novel interactive approach for approximate isocontouring of high-order data. The method is based on a two-phase hybrid rendering algorithm. In the first phase, coarsely seeded particles are guided by the gradient of the field for obtaining an initial sampling of the isosurface in object space. The second phase performs ray casting in the image space neighborhood of the initial samples. Since the neighborhood is small, the initial guesses tend to be close to the isosurface, leading to accelerated root finding and thus efficient rendering. The object space phase affects the density of the coarse samples on the isosurface, which can lead to holes in the final rendering and overdraw. Thus, we also propose a heuristic, based on dynamical systems theory, that adapts the neighborhood of the seeds in order to obtain a better coverage of the surface. Results for datasets from computational fluid dynamics are shown and performance measurements for our GPU implementation are given.

1998 ACM Subject Classification I.3.3 [Computer Graphics]: Picture/Image Generation–Display algorithms; I.3.1 [Computer Graphics]: Hardware Architecture–Graphics processors;

Keywords and phrases High-order finite elements, isosurface visualization, GPU.

Digital Object Identifier 10.4230/DFU.Vol2.SciViz.2011.276

1 Introduction

Computational simulations use finite element methods to approximate the solution of partial differential equations. While the use of low-order linear basis functions within the finite elements was common in the past, the increasing need for numerical accuracy and efficiency motivates the development of more sophisticated numerical methods. An example is the *hp*-adaptive finite element method used in computational fluid dynamics, which produces high-order approximations inside each element. This leads to more accurate representations of physical phenomena. However, for visual inspection, the rendering of such data can become costly due to expensive evaluation. Approximate representation is one way to offer a viable visualization approach, allowing a trade-off between rendering speed and accuracy. Low-order resampling and isocontouring algorithms such as Marching Cubes (MC) [8] are among the simplest solutions to this problem. Adaptive sampling variations of MC can further reduce the error and capture more complex structures [14, 17]. However, resampling approaches

Scientific Visualization: Interactions, Features, Metaphors. *Dagstuhl Follow-Ups, Vol. 2.*
Editor: Hans Hagen; pp. 276–291

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz Zentrum für, Germany

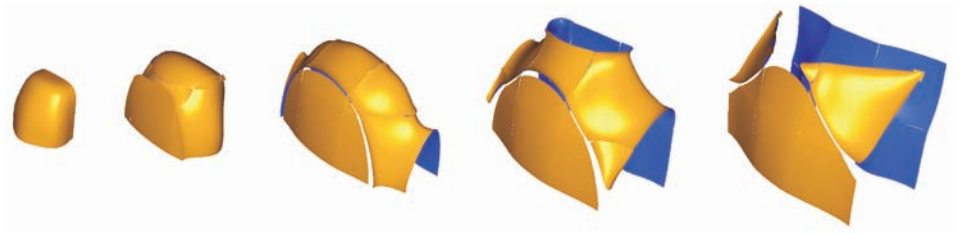


Figure 1 Isocontouring of a sequence of different density levels for a high-order dataset obtained from a Discontinuous Galerkin flow simulation.

introduce error and often lead to increased memory demands due to the large number of lower-order elements needed to represent the original data with sufficient accuracy.

To overcome these limitations, more attention has been paid recently to methods for direct contouring of high-order data. Usually, in these methods, isocontouring is realized as a root finding or gradient descent problem. Due to the high-order nature of the data, there is no closed-form solution for these approaches, and numerical methods must be used instead. Since these methods are typically computationally expensive, they are often carried out in a pre-processing step, which was applied in several mesh-extraction [14, 16, 17] and point-based algorithms [3, 20, 18, 10]. Although interactive rendering is possible with these techniques because the isosurface is computed during pre-processing, changing isovalues requires recomputation of the isosurface and hence the overall visualization is often no longer interactive. The ability to change isovalues interactively is present in some ray casting or ray tracing isocontouring algorithms [19, 11, 6]. However, the evaluation during rendering of such numerical methods typically leads to lower frame rates.

In this work we present a technique for the interactive approximate contouring of high-order data. It allows to choose the trade-off between rendering speed and rendering quality and is called iHOS (isocontouring of High-Order Surfaces) in this paper. It relies on a hybrid algorithm that distributes the isocontouring workload over pre-processing and rendering phases, allowing for interactive exploration by changing isovalues. The main components include:

- A parallel particle-based technique to generate a coarse, view-independent sampling of the isocontour in object space, which is only executed when the isovalue changes.
- An algorithm based on dynamical systems theory that allows for improved isosurface sampling.
- GPU-based ray casting that shoots a collection of rays for each quad around each sample in object space.

The method is designed to be efficiently mapped to parallel architectures and does not rely on inter-element connectivity information. Element reordering is not necessary for correct rendering, although performance improvement is achieved by processing elements in front-to-back order. Results are presented for datasets composed by convex cells. However, since no cross-cell rays are used, the method can be easily adapted to work with concave cells through the use of a more general point-in-polyhedron test. These are desired features that allow easy handling of structured and unstructured meshes as well. Figure 1 presents the results obtained with our method for a sequence of different isovalues for a high-order dataset obtained from a Discontinuous Galerkin flow simulation.

2 Related Work

There is a vast literature on isocontouring; the discussion below focuses on techniques for high-order data. Figueiredo *et al.* [3] proposed physically-based approaches for extracting triangle meshes from implicit surfaces. One method is particle-based while the second is based on a mass-spring system, and represents the first attempt on using particles to sample high-order data. This work inspired Witkin and Heckbert [20] to develop a point-based tool for the modeling and visualization of implicit surfaces. When used as a modeling tool, points represent handles for changing shapes. As a visualization tool, points are projected onto the surface and rendered as discs, with size and distribution adaptively computed according to the curvature of the surface. Meyer *et al.* [10] proposed a technique for isosurfaces generated from high-order finite element simulations that builds upon the approach of Witkin and Heckbert. Potential functions are used for particles repulsion, giving more control over particle distribution. This method handles cells with curved surfaces and allows for interactive visualization of a given isosurface. However, interactive exploration of different isovalues is not viable since every isovalue change typically takes several minutes due to resampling.

Kooten *et al.* [18] presented the first interactive particle-based method for implicit surface visualization that runs entirely on the GPU. The projection method is an adaptation of that by Witkin and Heckbert. Particle repulsion relies on a spatial-hash data structure that requires costly and frequent updates, thus preventing its use for rendering large datasets. Although not directly related to isocontouring, we mention the work by Zhou and Garland [21], which presents a point-based approach for the direct volume rendering of high-order tetrahedral data. Despite their effectiveness, point-based methods typically only provide a coarse representation for the isosurface, and accurate representations require a huge number of points. Haasdonk *et al.* [4] presented a multi-resolution rendering method for *hp*-adaptive data based on polynomial textures targeted to 2D rendering. Schroeder *et al.* [16] presented a mesh extraction method that explores critical points of basis functions to provide topological guarantees on the extracted mesh. Similar to other mesh extraction methods, it employs computationally expensive pre-processing to allow interactive exploration of arbitrary isovalues. Remacle *et al.* [14] proposed a method for resampling high-order data inspired by adaptive mesh refinement (AMR) methods. The original high-order data is down-sampled to a lower-order representation which is suitable for lower-order visualization algorithms like Marching Cubes [8]. The resampling error threshold can be adjusted, but low thresholds can lead to memory consumption explosion.

A competing strategy is to avoid pre-computation and compute isocontours during rendering. Nelson and Kirby [11] presented a ray tracing-based method for the isocontouring of spectral/*hp*-adaptive data. The method works by projecting the high-order functions onto each traced ray and computing the intersection with the isosurface from the resulting univariate function. Visualization error is carefully quantified and reduced. On the other hand, it typically takes several seconds to generate the final image, prohibiting interactive exploration of the data.

Knoll *et al.* [6] presented an interactive interval arithmetic-based method for the ray tracing of implicit surfaces on the GPU. It is currently one of the fastest ray tracers for implicits, presenting also robustness with respect to the root-finding process. However, the equations of the implicits must be converted to an inclusion-computable form, which increases the number of arithmetic operations needed to evaluate the equations. This is not a problem in the case of polynomials with only few coefficients, as can be verified by their

results. However, in the case of simulation data we usually have polynomials with hundreds of coefficients and thousands of polynomials to be evaluated for each frame, and the use of interval arithmetic may impact performance.

3 Isocontouring High-Order Surfaces

In this section we describe the details of the isocontouring High-Order Surface (iHOS) algorithm. We start by first giving an overall description, followed by details of each step.

3.1 iHOS design

Interactive rendering rates in iHOS are obtained by dividing the isocontouring workload between object and image space computations, each defined in a separate phase. The first phase, in object-space, generates an initial sampling of the isosurface by projecting particles (here called *seeds*) along the gradient field onto the surface. Only seeds successfully projected onto the surface are considered for further processing. Each projected seed generates a surface-tangent, seed-centered quadrilateral (quad) that covers the neighborhood of the seed. The second step uses the fragments generated by the rasterization of those quads as the initial points for a ray casting that refines the isosurface representation in image space. Figure 2 gives an overview of the iHOS pipeline.

To compute the initial isosurface approximation, we start by placing a set of seeds uniformly distributed inside each cell and projecting them onto the surface. The projection affects the density of the seeds and thus the sampling quality. There are methods described in the literature that handle this problem by using repulsion/attraction or birth/kill of seeds in a pre-processing stage [20, 10, 18]. Despite their effectiveness, these procedures are computationally expensive, requiring the update of complex data structures that are hard to efficiently map to parallel architectures. Although our experimental results show that satisfactory quality images are obtained even without handling the sampling problem, we additionally propose a heuristic that helps in reducing the artifacts of the final image. This heuristic is implemented as a pre-processing stage that analyzes, under the dynamical systems theory perspective, the seeds behavior along the gradient field during projection. Differently from the previous methods that must compute the pre-processing for each isovalue, this pre-computation is executed only once for the entire dataset and does not depend on a specific isovalue. The next sections give an in depth explanation of each step of our algorithm. We start by first reviewing the finite element (FE) high-order data subject to this work.

3.2 *hp*-adaptive FE Data

FE methods imply discretization. The physical domain Ω is discretized in a compatible mesh (meaning that the elements intersect only at the boundary faces, edges, or vertices). A given mesh M with N elements [14] can be defined as

$$M = \bigcup_{i=1}^{N} e_i \tag{1}$$

where e_i is the i-th element. In some FE method simulations, results are stored at the vertices of regularly distributed elements. However, recent numerical methods brought the concepts of h- and p-adaptiveness. h-adaptive methods vary the element size (h-refinement) while p-adaptive methods are those that can vary the order of the basis functions (p-refinement). The hp-adaptive methods combine both. These methods are of special interest because they

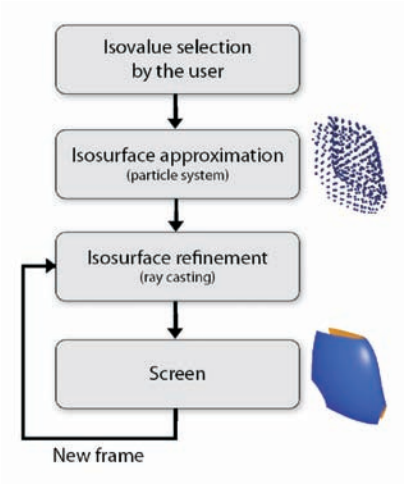


Figure 2 iHOS pipeline: An initial sampling of the isosurface is generated in object space by projecting coarsely seeded particles over the isosurface along the gradient field in object space. Each projected point generates a quadrilateral whose fragments will be used as starting points for the ray casting in the image-based phase.

can, through the balance between h- and p-adaptiveness, increase the convergence order of computations. In p or hp methods, results are defined continuously for each element using high-order functions.

In the specific case of Discontinuous Galerkin methods [13] the solution is computed separately for each element, which means that the solution can be discontinuous at the element boundaries. It is also common to have the higher-order basis functions defined in a reference space, and a mapping function that transforms coordinates from reference to world space. Although this mapping can be non-linear, we currently only handle transformations that include rotation, translation, reflection, and scaling.

3.3 Object Space Sampling

The task of finding an isosurface for a given polynomial data can be formulated as a root-finding problem. For high-order polynomials, the lack of closed form solutions lead to the use of iterative numerical methods. Newton-Raphson (NR) is one of the best known methods for numerical root finding. Additionally, it presents good convergence rates when the starting point is already close to the desired solution. However, uniformly distributed seeds may not be close to the solution and NR may fail, leading to poor sampling. One possible solution is to guide the seeds through the gradient field until they get closer to the isosurface and then apply NR to refine the projection. Since we just want to bring the seeds closer to the isosurface, we decided for guiding the points by means of integration. After the integration step it is assumed that the seeds are closer to the desired isosurface and we start with the NR iterations. Directional NR [7] along the gradient is used, since it has the additional advantage of quadratic convergence. Equation 2 shows the formulation used for the NR iterations.

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \frac{f(\mathbf{x}^k)}{\nabla f(\mathbf{x}^k) \cdot \nabla f(\mathbf{x}^k)} \nabla f(\mathbf{x}^k) \tag{2}$$

where k is the time step ($k = 0, 1, \ldots$), $\mathbf{x}$ is the point position in reference space, and f is the polynomial for the current cell.

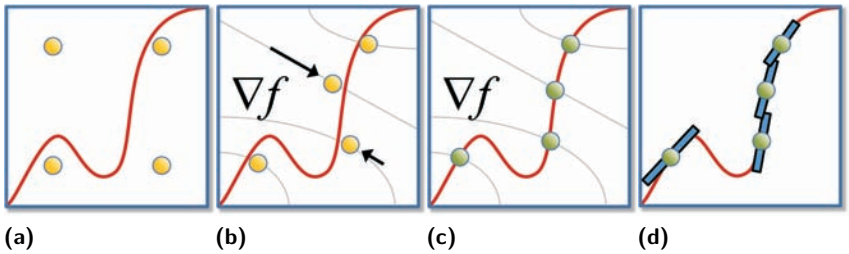


Figure 3 Isosurface approximation: (a) Seeds are initially distributed uniformly into each cell. (b) Seeds are guided through integration along the gradient field towards the desired isosurface. (c) Seeds are projected using directional NR on the isosurface. (d) Surface-tangent quads are generated for each point projected onto the isosurface.

All computations are performed separately for each seed point since no neighborhood information is needed. Figure 3 gives an overview of the first phase.

3.4 Image Space Refinement

The input for the second phase of our method is a set of surface-tangent, seed-centered quads that together cover the isosurface. Points lying on these quads can be used as starting points for the ray casting that will refine the representation of the isosurface. These points are efficiently obtained through the rasterization of the quads in image space. The fact that these points are already close to the isosurface implies that we can use the directional NR method along the ray direction, as show in Figure 4. Equation 3 shows the formulation for the directional NR used for the fragment projection:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \frac{f(\mathbf{x}^k)}{\nabla f(\mathbf{x}^k) \cdot \mathbf{r}} \mathbf{r}\,, \tag{3}$$

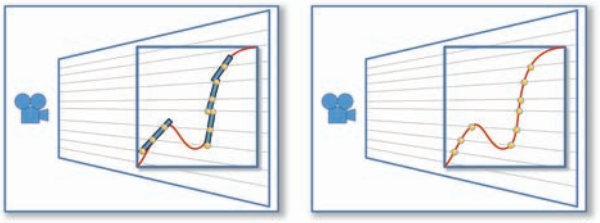
where k is the time step ($k = 0, 1, \ldots$), $\mathbf{x}$ is the point position in reference space, f is the polynomial for the current cell, and $\mathbf{r}$ is the ray vector in reference space.

The generated ray and fragment coordinates are transformed into the reference coordinate system, where the ray-isosurface intersection computations are performed. After the maximum number of iterations is reached, if the fragment is not projected onto the isosurface (within an error tolerance bound) or it has been moved outside of the cell, it is discarded. If the fragment is successfully projected, shading is computed and the corresponding pixel color is updated.

4 Sampling Strategies

Both steps, the object space sampling (Section 3.3) and the image-based refinement (Section 3.4), require appropriate sampling for obtaining a complete isosurface representation. In some cases holes can result due to improper sampling. As a limit case of the object space sampling, infinitely dense seeding would guarantee a complete isosurface representation, but in practice, no guarantees about the resulting isosurface sampling can be given.

The reason is that in general no assumptions about the behavior of the gradient field can be made, and hence of the mapping of the seeds to the isosurface. It is, for example, generally not possible to estimate the maximum distance between two seeds when they are mapped to an isosurface of arbitrary isolevel. A limit case of the image space refinement is a single



(a) (b)

Figure 4 Isosurface refinement: (a) Quads are rasterized in image space. (b) Each fragment is projected against the isosurface through directional Newton-Raphson.

quad that spans the complete screen, which would be equivalent to a ray casting approach. This would lead to the typically low performance of these methods due to computationally expensive root finding along the viewing rays. Our method operates between the two mentioned limit cases. To motivate the parametrization and a sampling technique, we will first discuss the relevant issues and implications that arise in the context of sampling.

4.1 Sampling Issues

The proposed approach suffers from the fact that the fragments are processed independently and therefore it is impossible to detect, and to directly fix holes in the isosurface. Holes can be caused by different reasons:

1. the root finding may fail, e.g. due to insufficient iteration count,
2. the root finding may deliver a point on a nearby isosurface if the quad is not well aligned with the isosurface, i.e. the isosurface exhibits too high curvature with respect to the initial sampling,
3. the quad sizes are insufficient.

The first cause is hard to address explicitly because the efficiency of our method relies on synchronized root finding and hence constant iteration count (see Section 5.2). However, this reason is typically negligible. Because we use quads that are aligned with the tangent plane of the isosurface at each sampling seed, the second case depends only on the quality of the initial sampling and the last one on the quad sizes. The initial sampling has to be performed each time the isolevel is changed and therefore the exploration of isocontours requires high performance for this operation. This makes the use of expensive approaches not feasible for this step. Unfortunately, it is generally not possible to find a distribution of the seeds for the object space sampling that leads to uniformly sampled isosurfaces for all possible isovalues. This makes non-uniform quad sizes for the image space refinement necessary.

Therefore, we propose a method for estimating sufficient quad sizes in Section 4.2 and place the seeds for the object space sampling regularly inside each cell. One problem is that even comparably large quads cannot guarantee hole-free coverage of the isosurface because the sampling points can be arbitrarily distant due to the possible intricacy of the gradient field. Additionally, large quads typically lead to high overdraw and hence to lowered performance. All in all, a balance has to be found between the number of initial seeds for the object space sampling and the sizes of the quads.

For approximate results at high performance that can serve as a preview to more expensive but robust methods such as that by Knoll *et al.* [6], the sampling in object space can be

parametrized and the size for the quads can be derived from that sampling as shown in Section 6. As an alternative, we now propose a method to estimate conservative quad sizes for individual quads.

4.2 Estimating Quad Sizes using FTLE

A method to determine quad sizes that guarantees a complete covering of the isosurfaces under the assumption of an appropriate object-space sampling can be derived from dynamical systems theory. The finite-time (or local) Lyapunov exponent (FTLE) was originally defined to measure the predictability of dynamical systems [9, 5]. More precisely, it measures the exponential growth that a perturbation undergoes when it is advected for a finite time interval in a vector field. The flow map $\boldsymbol{\phi}^T(\mathbf{x})$, which maps a sample point $\mathbf{x}$ to its advected position $\boldsymbol{\phi}^T(\mathbf{x})$ after advection time T, is the basis for the Cauchy-Green deformation tensor $\nabla\boldsymbol{\phi}^T(\mathbf{x})$. Using

$$\Delta^T(\mathbf{x}) = (\nabla\boldsymbol{\phi}^T(\mathbf{x}))^\top \cdot \nabla\boldsymbol{\phi}^T(\mathbf{x}) \tag{4}$$

and $\lambda_{\max}$ being the largest eigenvalue, leads to the finite-time Lyapunov exponent [5]:

$$\sigma^T(\mathbf{x}) = \frac{1}{|T|} \ln \sqrt{\lambda_{\max}(\Delta^T(\mathbf{x}))}\,. \tag{5}$$

For our application, we are only interested in the growth of the distance between two seeds as they are guided by the gradient field and therefore we omit the normalization by advection time and the logarithm in Eq. 5. Additionally, since we look for points on the isosurfaces, i.e. the intersections of isosurfaces with gradient field lines starting at the seeds, the integration along the field lines shall not be limited by integration time (or length). Instead, it has to be limited by the prescribed isolevel l. We define $\boldsymbol{\phi}^l(\mathbf{x})$ to be mapping the position $\mathbf{x}$ to the isosurface of isolevel l along the corresponding gradient field line (the field line is stopped if a critical point of the gradient field is reached). Using Eq. 5, this would lead to the separation factor

$$s^l(\mathbf{x}) = \sqrt{\lambda_{\max}(\Delta^l(\mathbf{x}))}\,. \tag{6}$$

We follow a direct approach for evaluating $s^l(\mathbf{x})$. To avoid numerical issues and to be able to exclude certain seeding neighbors $\mathbf{n} \in \mathcal{N}(\mathbf{x})$ of $\mathbf{x}$, as described below, the computation of $s^l(\mathbf{x})$ is not based on the gradient of $\boldsymbol{\phi}^l(\mathbf{x})$, but it is calculated directly:

$$s^l(\mathbf{x}) = \max_{\mathbf{n}\in\mathcal{N}(\mathbf{x})} \frac{||\boldsymbol{\phi}^l(\mathbf{n}) - \boldsymbol{\phi}^l(\mathbf{x})||}{||\mathbf{n}-\mathbf{x}||}\,. \tag{7}$$

The quad size $d(\mathbf{x})$ at $\boldsymbol{\phi}^l(\mathbf{x})$ is $s^l(\mathbf{x})$ times the corresponding seeding distance $||\mathbf{n}-\mathbf{x}||$. If the resolution of the object space sampling is appropriate, using these quad sizes guarantees a complete covering of the isosurface at level l. For obtaining quad sizes that are appropriate for any isolevel, the maximum separation $\hat{s}(\mathbf{x})$ over all isolevels, computed in a preprocessing step, is used to determine the quad sizes:

$$\hat{s}(\mathbf{x}) = \max_{l_{\min}\le l\le l_{\max}} s^l(\mathbf{x})\,. \tag{8}$$

This quantity relates to the FTLE maximum by Sadlo and Peikert [15]. Unfortunately, neighboring seeds can get mapped to different isosurface parts, making the resulting quad sizes

too conservative. This leads to unnecessary overdraw during rendering and hence lowers the overall performance. As a remedy, we propose a heuristic that reduces the overdraw caused by these cases. The idea is to exclude neighbors of $\mathbf{x}$ from the computation of $\hat{s}(\mathbf{x})$, that are not located in its vicinity on the same isosurface. There are several possible approaches for detecting if two points are adjacent on the same isosurface. The straightforward approach would be to compute the geodesic distance between the points along the isosurface. However, this is considered impractical and too expensive.

A criterion for testing if a mapped point $\boldsymbol{\phi}^l(\mathbf{x})$ and its mapped neighbors $\boldsymbol{\phi}^l(\mathbf{n}), \mathbf{n} \in \mathcal{N}(\mathbf{x})$ are located close to each other on the same isosurface can be motivated by the fact that the gradient is always aligned with the isosurface normal. To test if a point and its neighbor are located on the same isosurface, the behavior of the gradient field is analyzed on the straight line connecting them. If the isosurface is planar between the two points, the line follows the isosurface and intersects the gradient field everywhere perpendicularly. If, on the other hand, the isosurface is non-planar between the points or if the points even lie on different isosurface regions, there are positions on the segment where the gradient is not perpendicular. We introduce the following measure $a_{\min}$ for the minimum angle between the gradient and the segment $\mathbf{s}_n$ going from $\mathbf{x}$ to $\mathbf{n}$:

$$a_{\min} = \max_{t \in [0,1]} |\mathbf{s}_n \cdot \nabla s(\mathbf{x} + t\mathbf{s}_n)| / (||\mathbf{s}_n|| ||\nabla s(\mathbf{x} + t\mathbf{s}_n)||) \,. \tag{9}$$

Then, the neighbor $\mathbf{n}$ is excluded from the computation if $a_{\min}$ exceeds a user-defined threshold. In practice, this threshold can be typically set to 0.5 for suppressing most of the erroneous neighbors in order to prevent too conservative quad sizes and reduce the performance loss due to overdraw.

5 Implementation

The design of iHOS was to take full advantage of parallel architectures. Although our current implementation runs on a single GPU, it could be easily distributed over a GPU cluster. We start by first describing the data structures used. Afterwards we go into the details of the implementation.

5.1 Data Storage

For the early cell culling we use an interval-tree [2] that stores the extrema of the scalar field for each cell. This tree is stored and processed on the CPU side. The remaining data structures are stored on the GPU through the use of different buffers. Each time the user selects a different isovalue, we need to load the initial set of uniformly distributed seeds and compute the isosurface approximation. Thus we allocate two seed arrays on the GPU: one with the initial unprojected seeds and another to store the projected seeds. For each unprojected seed, its corresponding FTLE value is stored as its w coordinate. Actually, those seed arrays are implemented as vertex buffer objects (VBOs). We refer to VBO_{unproj} for the VBO with unprojected seeds and VBO_{quads} for the VBO that stores the quads. Polynomial, gradient, and cell boundary data are read-only and are made available for GLSL through the use of bindable uniform buffer objects (BUBOs). Since BUBOs are limited in size (usually 64KB) we chose to create a separate BUBO for each cell and to bind them on demand.

5.2 GLSL Shaders

Both phases of the algorithm are implemented through a set of shader programs. The first step of the algorithm is implemented through a GLSL program composed of a vertex shader and a geometry shader. The vertex shader reads the VBO_{unproj} and projects the seeds onto the isosurface (Section 3.3). These seeds are streamed up to the geometry shader and used to generate surface tangent quads. The quads are resized accordingly to the FTLE value of the seed (Section 5.1). In order to record the generated quads into the VBO_{quads} we interrupt the pipeline just after the geometry shader through the use an OpenGL extension called *Transform Feedback* (or stream-out in DirectX terminology). The GLSL program responsible for the second step is composed of a vertex and a fragment shader. The vertex shader just redirects the vertices of the quads to the rasterization stage. The fragment shader implements the core of the second step. It performs a ray casting using the input fragment as the starting point for the root-finding iterations (Section 3.4). As said before, fragments that are not sucessfully projected are discarded. However, it must be observed that fragments successfully projected onto the isosurface are only shaded according to the new position, without having their depth coordinate updated. This procedure greatly increases method's performance by allowing the use of the early-z test. A side effect of this approach is that some noise can appear on some regions of the isosurface. This happens because a fragment closer to the camera can be projected on a far surface, being thus wrongly shaded. This effect is greatly reduced through the comparison between the angles of the polynomial gradient vectors at the fragment positions before and after projection. If the two angles differ above a certain value, it is assumed that the fragment jumped to another surface. In this case one have just to discard the fragment. In both steps, the shaders must compute several evaluations of the polynomials and their gradients. A shader capable of evaluating a general polynomial with an arbitrary degree should contain a loop that could not be unrolled by the compiler. To increase performance, we decided to keep several versions of these shaders, each one targeted to a specific polynomial degree. For the polynomial evaluation itself we use static expressions generated by a multivariate Horner scheme approach [1]. We decided to keep the number of integration and NR iterations static in order to reinforce thread synchronization, increasing performance.

5.3 Computation Flow

When the user chooses a desired isovalue the cells containing the isovalue are selected. These cells are inserted in a separate list and arranged in front-to-back order through fast sorting that does not have to be exact. This ordering is not necessary, but it helps the second step of the algorithm to avoid processing occluded fragments, increasing performance. The shaders corresponding to the polynomial degree of the current cell are activated. The corresponding BUBO (containing the polynomial, gradient, and boundary data) for the current cell is bound and the seeds are projected. After all seeds are projected, we start with the second step. Again, the cell list is traversed, and now the ray casting shaders and corresponding BUBOs are bound. Fragments successfully projected onto the isosurface are recorded in the frame buffer.

6 Results

Results and performance measurements were obtained on a computer with an Intel Core 2 Quad 2.4 GHz processor, with 4 GB of RAM, Linux operating system, and NVidia GeForce

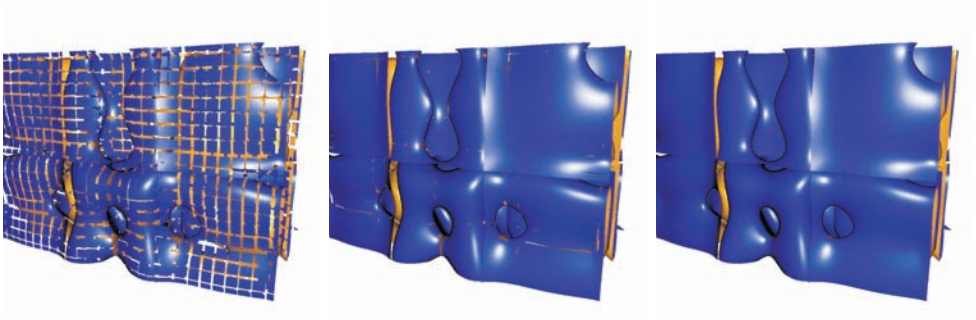


Figure 5 Renderings of isovalue ≈ 2 with varying q_s. From left to right, the s_q value and corresponding frames per second (fps) are: $s_q = 0.75$ and 45 fps, $s_q = 1.0$ and 39 fps, $s_q = 1.25$ and 29 fps.

8800 GTX video card. To demonstrate the method we used synthetic data and two density datasets generated by *hp*-adaptive Discontinuous Galerkin flow simulations. The scalar fields of all datasets are defined analytically by a multivariate polynomial defined in the reference space in a monomial basis of the form:

$$P(\mathbf{x}) = \sum_{i+j+k<=n} c_{ijk} x^i y^j z^k \tag{10}$$

with i, j, $k \in \mathbb{N}_0$, the maximum order n, the coefficients c_{ijk}, and the reference space coordinates x, y, z. The mapping functions for all datasets include only translations. The smaller dataset is composed of 119 hexahedral cells with polynomial degrees varying between 5 and 6. The larger dataset is composed of 34,535 convex polyhedral cells (tetrahedra, prisms, and hexahedra) with a scalar field defined by degree 3 polynomials.

6.1 Performance

For each phase of the algorithm a set of parameters can be adjusted. For the first phase these parameters are the number of initial seeds (n_s), number of integration steps (n_i), number of NR steps (n_{N1}), and the error for the NR projections (ϵ_1). For the second phase the parameters that can be adjusted are the number of NR steps (n_{N2}), the scale factor for the quad size (s_q), and the error for the NR projections (ϵ_2). Although the great number of parameters may be confusing at first, they offer great flexibility to the user, allowing the tuning towards accuracy or interactivity. From our experience we observed also that in general one can explore both accuracy and performance by adjusting just a few of them, keeping the others fixed. For all presented measurements we used screen resolutions of 1024^2 pixels.

All renderings were made with active early cell culling and with front-to-back ordering for the cells. Figure 5 shows performance measurements for the small dataset. The fixed parameters, and their respective values, are: $n_s = 10^3$, $n_i = 50$, $n_{N1} = 10$, $\epsilon_1 = 10^{-3}$, $n_{N2} = 3$ and $\epsilon_2 = 10^{-4}$. These values were chosen through trial-and-error. The only variable parameter is s_q. Due to early cell culling only 16 cells ($\approx 7.43\%$ of the total) were processed. The results of Figure 6 show performance measurements for the bigger dataset. The fixed parameters are: $n_s = 8^3$, $n_i = 50$, $n_{N1} = 10$, $\epsilon_1 = 10^{-3}$, $n_{N2} = 3$ and $\epsilon_2 = 10^{-4}$. These values were also chosen through trial-and-error. Again, the only variable parameter is s_q. Due to early cell culling 3,781 cells ($\approx 9.13\%$ of the total) were processed.

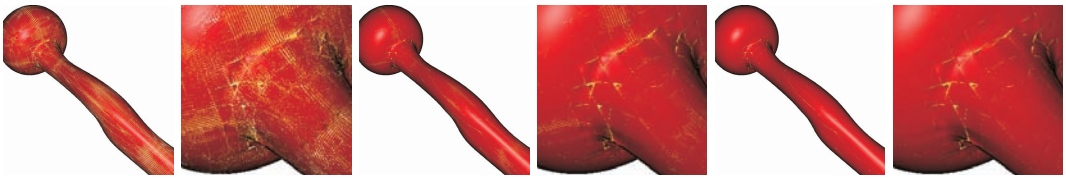


Figure 6 Renderings of isovalue ≈ 0.9983 with varying s_q. From left to right, rendering and zoomed images with the following s_q value and corresponding frames per second (fps): $s_q = 0.5$ and 21 fps, $s_q = 0.75$ and 14 fps, $s_q = 1.0$ and 12.5 fps.

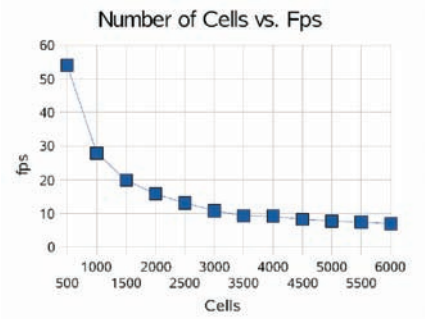


Figure 7 Performance vs. number of cells. The non-linearity can be explained by the use of the early-z test, which avoids the processing of occluded fragments.

Figure 7 shows how the method scales with respect to the number of cells. These measurements were made with degree 3 polynomials extracted from the bigger dataset. It can be seen that the rendering cost does not grow linearly. This can be explained by the use of the early-z test, which avoids the processing of fragments or even cells that are totally occluded. This behavior is reinforced since we use front-to-back ordering for the cells. We may still have some overdraw since we do not order the quads inside the cell. However, from our experiments, this has not affected significantly the performance in the general case.

The method proposed by Knoll *et al.* [6] may look, at first, as a competing approach. Despite the fact that their method is meant for the robust rendering of implicits, it is fast and could be applied separately to each cell of our datasets. Figure 8 shows a comparison between our method and Knoll's for the rendering of a single cell of our small dataset. Visual inspection shows that our method is able to achieve higher framerates at better image quality. One reason for the difference in performance is probably related to the extra cost involved with the use of inclusion arithmetic in Knoll's method. Although the use of inclusion arithmetic guarantees error bounds for the results, it tends to produce inferior results if larger error bounds are used. Since we are not primarily interested in the control of accuracy, we rely on the direct evaluation of the original polynomial representation, which is computationally cheaper and thus can be used in a previewing system.

Regarding the time spent to recompute the object space approximation at each isovalue change, for all our tests, they took less than 2 seconds. In the case of the smaller dataset, it takes less than 1 second to recompute the seed projection. All measurements were made with 50 integration iterations and 10 NR iterations.

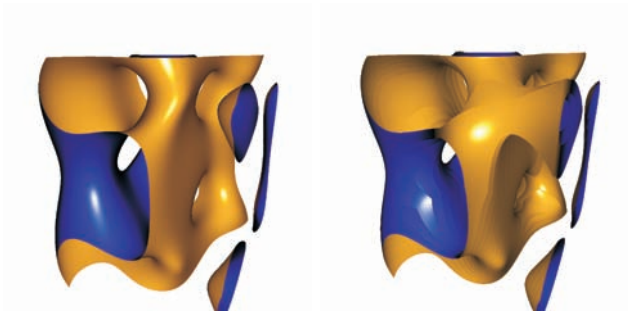


Figure 8 Comparison with Knoll's method. (left) Polynomial of degree 5 rendered by our method at ≈ 30 fps. With the standard settings (depth $= 10$ and error $= 0$), Knoll's method rendered this image at ≈ 8 fps without visible artifacts. (right) After optimization of their parameters (depth $= 4$ and error $= 0$) to reach the best possible image quality at ≈ 30 fps. Both images rendered in a 1024^2 pixel screen.

6.2 Isocontouring Quality

As a previewing system the proposed method does not focus on high accuracy. Nevertheless, it is capable of delivering results of reasonable quality. Figure 9 shows a comparison between images generated by our method and the same images generated by POV Ray, a well known, tested, and freely available ray tracer [12]. The discontinuities observed in these renderings are due the Discontinuous Galerkin method used to compute the simulations. Our method handles continuous data properly, as can be seen in Figure 10.

Despite the reasonable quality of the images, one may want to have better visual quality for the isocontouring. In such cases the user can switch to a robust isocontouring system, as the one proposed by Knoll *et al.* [6]. However, we have also developed some heuristics in order to allow our method to generate better quality images. The quality of the final sampling depends on the relation between the number of seeds and the size of the quads. The exact relation between these two parameters is difficult to assess since the projection changes seed density. The FTLE-based heuristic tries to estimate "optimal" quad sizes that

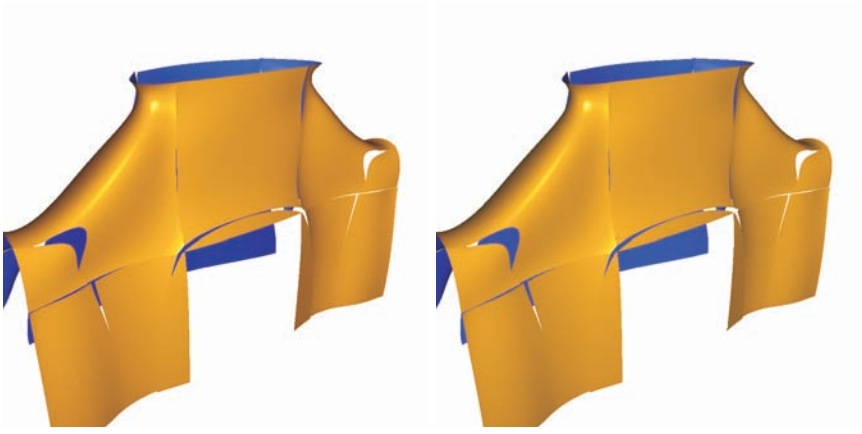


Figure 9 Comparison against POV Ray. (left) Image generated with our method, with the following settings: $n_s = 10^3$, $n_i = 50$, $n_{N1} = 10$, $\epsilon_1 = 10^{-3}$, $n_{N2} = 3$, $\epsilon_2 = 10^{-4}$, $s_q = 1.5$ and isovalue ≈ 5.8437. This scene was rendered at 8.7fps. (right) Reference image generated from the same dataset and same isovalue with POV Ray. Both images rendered at 1024^2 pixels.

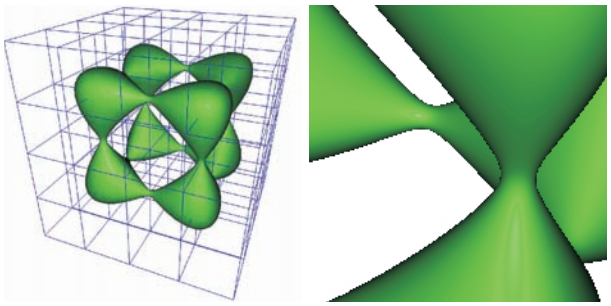


Figure 10 Rendering of Tangle dataset sampled with a grid of $4^3 = 64$ cells (left). The dataset is continuous inside and across cells. Zoomed image (right) shows how our method handles the continuity at the borders correctly. Settings for this rendering are: $n_s = 10^3$, $n_i = 50$, $n_{N1} = 10$, $\epsilon_1 = 10^{-3}$, $n_{N2} = 3$, and $\epsilon_2 = 10^{-4}$. The isovalue is ≈ -1.98. Image was rendered at 1024^2 with ≈ 9.6 fps.

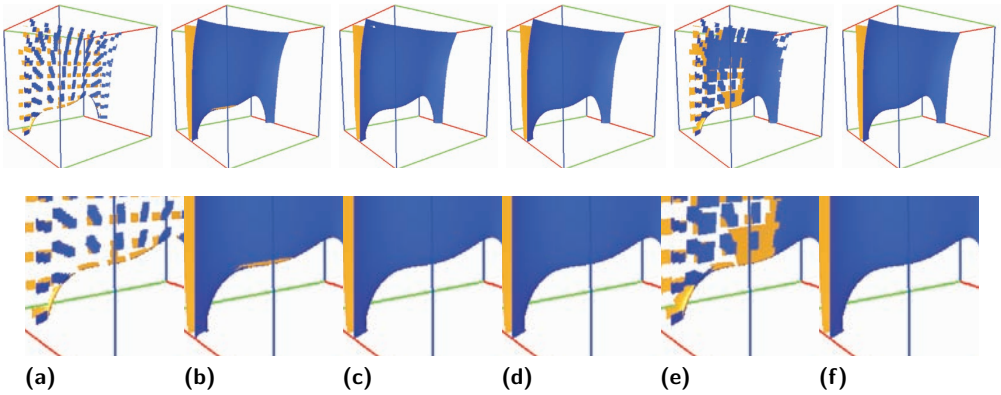


Figure 11 Sampling issues related to quad size: (a) Standard quads reduced to 30% of original size. (b) Sampling using standard quads: some artifacts can be seen due to poor sampling. (c) Quads scaled by the FTLE and reduced to 30% of their size. (d) Sampling obtained with quads resized by the FTLE approach. (e) Quads resized by our FTLE approach in conjunction with a heuristic that reduces its conservativeness, reduce in 30%. (f) Sampling obtained with our FTLE approach in conjunction with the heuristic.

"work" for all isovalues considering an initial seeding given by the user. Since the FTLE does not take into account the topology of the isosurface, it can be too conservative, by considering distances between seeds in disjoint surfaces. For this case we adopt a test that estimates whether points are in disjoint surfaces (Section 4.2). Figure 11 illustrates the sampling problems related to bad seeding/quad size relation and how our heuristic resizes the splats according to that seeding, reducing effectively the artifacts in the final image. It also illustrates the effect of our disjoint-surfaces test on reducing the FTLE conservativeness. Despite the fact that it improves the quality of the final rendering, the FTLE makes the method less attractive for previewing since it reduces its performance.

7 Conclusion and Future Work

This work has presented iHOS, an interactive approach for isocontouring higher-order data generated by *hp*-adaptive discretization methods. The method operates with high-order data whose mapping function includes only rotation, translation, reflection, and scaling. The

method does not rely on complex data structures and does not resample the original data into lower-order representations. Results are presented for datasets composed by convex cells. However, since no cross-cell rays are used, the method can be easily adapted to work with concave cells through the use of a more general point-in-polyhedron test. Although not meant to generate accurate images, the presented results show that general quality of the generated images is good. Several parameters can be adjusted in order to control the trade-off between efficiency and accuracy; it has been shown that one can explore both (accuracy and performance) by adjusting just a few parameters, keeping the others at fixed values. Some points should be addressed in future work: further reduction of the overdraw during quad rasterization, development of an improved initial seed distribution heuristic for the first phase, handling of large datasets, optimizations for subpixel cells and handling of non-linear mapping functions between reference and world spaces. For future work we also plan to adapt this method to handle dynamic datasets.

Acknowledgements

We thank our colleagues from the Institut für Aero- und Gasdynamik from Universität Stuttgart, Germany, for their continuous support and for providing datasets. We also thank Aaron Knoll for kindly providing code for performance tests and comparisons, and the anonymous reviewers for their comments and insightful suggestions. C.A. Pagot would like to acknowledge the support of CAPES-Brazil (Probral 3192/08-3) and CNPq-Brazil (140238/2007-7).

References

1 Martine Ceberio and Vladik Kreinovich. Greedy algorithms for optimizing multivariate Horner schemes. *ACM SIGSAM Bulletin*, 38(1):8–15, 2004.

2 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(2):158–170, 1997.

3 Luiz Henrique de Figueiredo, Jonas de Miranda Gomes, Demetri Terzopoulos, and Luiz Velho. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of Graphics Interface '92*, pages 250–257, 1992.

4 B. Haasdonk, M. Ohlberger, M. Rumpf, A. Schmidt, and K. G. Siebert. Multiresolution visualization of higher order adaptive finite element simulations. *Computing*, 70(3):181–204, 2003.

5 G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D*, 149(4):248–277, 2001.

6 Aaron Knoll, Younis Hijazi, Andrew Kensler, Mathias Schott, Charles D. Hansen, and Hans Hagen. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Computer Graphics Forum*, 28(1):26–40, 2009.

7 Yuri Levin and Adi Ben-Israel. Directional Newton methods in n variables. *Mathematics of Computation*, 71(237):251–262, 2002.

8 William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

9 E. N. Lorenz. A study of the predictability if a 28-variable atmospheric model. *Tellus*, 17:321–333, 1965.

10 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.

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

12 Povray. Persistence of Vision Raytracer. Website, 2009. `http://www.povray.org`.

13 W. Reed and T. Hill. Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

14 B. Remacle, N. Chevaugeon, É. Marchandise, and C. Geuzaine. Efficient visualization of high order finite elements. *International Journal for Numerical Methods in Engineering*, 69(4):750–771, 2006.

15 Filip Sadlo and Ronald Peikert. Efficient visualization of Lagrangian coherent structures by filtered AMR ridge extraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1456–1463, 2007.

16 William J. Schroeder, Francois Bertel, Mathieu Malaterre, David Thompson, Philippe P. Pebay, Robert O'Bara, and Saurabh Tendulkar. Framework for visualizing higher-order basis functions. In *Proceedings of the IEEE Visualization Conference*, pages 43–50, 2005.

17 William J. Schroeder, Francois Bertel, Mathieu Malaterre, David Thompson, Philippe P. Pebay, Robert O'Bara, and Saurabh Tendulkar. Methods and framework for visualizing higher-order finite elements. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):446–460, 2006.

18 Kees van Kooten, Gino van den Bergen, and Alex Telea. *GPU Gems 3*, chapter Point-based visualization of metaballs on a GPU. Addison-Wesley,Nguyen, Hubert, 2007.

19 David F. Wiley, Hank Childs, Bernd Hamann, and Kenneth I. Joy. Ray casting curved-quadratic elements. In *VISSYM'04: Proceedings of the Symposium on Data Visualisation*, pages 201–210, 2004.

20 Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of ACM SIGGRAPH 1994*, pages 269–277, 1994.

21 Yuan Zhou and Michael Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1229–1236, 2006.