Scalable Hybrid Unstructured and Structured Grid Raycasting

Philipp Muigg, Markus Hadwiger, Helmut Doleisch, Helwig Hauser

Abstract— This paper presents a scalable framework for real-time raycasting of large unstructured volumes that employs a hybrid bricking approach. It adaptively combines original unstructured bricks in important (*focus*) regions, with structured bricks that are resampled on demand in less important (*context*) regions. The basis of this focus+context approach is interactive specification of a scalar degree of interest (DOI) function. Thus, rendering always considers two volumes simultaneously: a scalar data volume, and the current DOI volume. The crucial problem of visibility sorting is solved by raycasting individual bricks and compositing in visibility order from front to back. In order to minimize visual errors at the grid boundary, it is always rendered accurately, even for resampled bricks. A variety of different rendering modes can be combined, including contour enhancement. A very important property of our approach is that it supports a variety of cell types natively, i.e., it is not constrained to tetrahedral grids, even when interpolation within cells is used. Moreover, our framework can handle multi-variate data, e.g., multiple scalar channels such as temperature or pressure, as well as time-dependent data. The combination of unstructured and structured bricks with different quality characteristics such as the type of interpolation or resampling resolution in conjunction with custom texture memory management yields a very scalable system.

Index Terms—Volume Rendering of Unstructured Grids, Focus+Context Techniques, Hardware-Assisted Volume Rendering

1 Introduction

Unstructured grids are an important volumetric representation that is especially common in the field of computational fluid dynamics (CFD), e.g., simulations of engineering problems computed with finite volume methods. Real-world simulation grids are often comprised of a variety of cell types, such as tetrahedra, hexahedra, octahedra, and prisms. However, most rendering approaches convert general unstructured grids into tetrahedral grids in a pre-processing stage and only handle tetrahedra during actual rendering. This subdivision of more complex cells into linear subcells, however, naturally prevents interpolation of a C^1-continuous function within non-tetrahedral cells. Thus, it is desirable for many real-world applications to avoid tetrahedralization for rendering, which is a major goal of our rendering method. Naturally, another challenge for rendering is the number of cells, which can easily be several hundred thousand or even millions of cells. Avoiding tetrahedralization also reduces the number of cells that need to be rendered significantly. Figure 2 shows a dataset of a generator that contains six million cells of mixed type (for one time step), which can be rendered interactively by our system.

The four main approaches for GPU-based rendering of unstructured grids are *cell projection* via the projected tetrahedra algorithm [22], *raycasting* [29], *resampling* into a structured grid followed by rendering this grid instead of the original unstructured grid [33], and *point-based approaches* [35]. One of the most important problems when rendering unstructured grids is to obtain a correct visibility order. Cell projection and point-based approaches require explicit visibility sorting to be performed, which is a major bottleneck of these methods. Raycasting approaches implicitly produce a correct visibility order, and thus do not need to perform explicit cell sorting at all. This property is a huge advantage of raycasting approaches and makes them competitive with cell projection when for the latter not only the time for projection and rendering but also for sorting is considered. Our framework employs raycasting as basic rendering method, but in contrast with similar approaches does not require the entire grid to be resident in GPU memory due to bricking.

Another important problem is how data interpolation is performed.

- *Philipp Muigg, Markus Hadwiger, and Helmut Doleisch are with the VRVis Research Center, Austria, E-mail: {muigg| msh| doleisch}@vrvis.at.*
- *Helwig Hauser is with the University of Bergen, Norway, E-mail: helwig.hauser@uib.no.*

Manuscript received 31 March 2007; accepted 1 August 2007; posted online 6 November 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

Although CFD simulations are commonly computed on a per-cell basis (*cell-centered data*), visualization with interpolation usually builds on data values given at the vertices of the grid (*vertex-centered data*). If necessary, conversion between these different representations can be performed. The main problem, however, is how to perform interpolation when rendering general unstructured cell types. Cell projection is usually restricted to tetrahedral cells and thus linear interpolation, i.e., barycentric interpolation within individual tetrahedra. An important goal of our work is to be able to render the original cells, and thus also to perform consistent interpolation in these cells. A powerful method for interpolating general polygons are *mean value coordinates* [8], which have been extended for closed triangular meshes [13] and for general polyhedra [17]. Our method offers different interpolation options and builds on mean value coordinates for high-quality interpolation in general cells.

The approach presented in this paper renders large, and possibly time-dependent, unstructured grids in real-time by performing raycasting through a hybrid unstructured and structured brick subdivision of the original unstructured grid. We employ a *focus+context* approach where the goal is to render as many bricks as possible using the original cells, especially bricks in the *focus*, and render less important bricks in the *context* using a structured grid obtained via on-the-fly resampling. Focus and context regions are selected via interactive specification of a degree of interest (DOI) function [5]. DOI values in $[0,1]$ are specified for each grid vertex or cell, and constitute an additional volume that is always used during rendering in addition to measured or simulated data values such as temperature or pressure. That is, our system handles multi-variate scalar data and at any one time renders one selected scalar channel such as temperature in conjunction with the scalar DOI function. The original grid cells are never subdivided into tetrahedra, but are rendered directly using one of several options for data interpolation. Scalability is mainly achieved by combining:

- Unstructured and structured bricks (hybrid rendering)
- Different structured brick resolutions
- Different interpolation options and adaptive sampling rate
- Distinguishing between focus and context regions
- Custom dynamic texture memory management

Furthermore, although we are focusing on rendering on a single GPU, our method would also be easily parallelizable by distributing individual bricks to multiple GPUs [25].

To summarize, the major contributions of this paper are:

- Hybrid raycasting through unstructured and structured bricks
- On-demand resampling steered by interactive focus+context specification
- Rendering of original cell types with interpolation

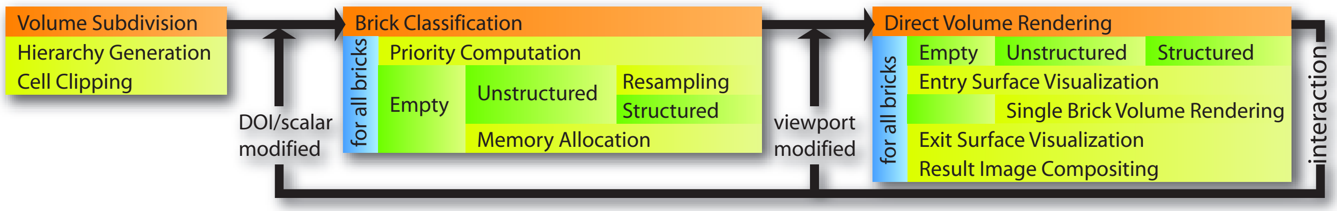

Fig. 1. Overview of our hybrid raycasting pipeline. In a pre-processing step, the grid is subdivided into a kD-tree hierarchy of bricks. Given the current *degree of interest* (DOI) function, bricks are classified for rendering as unstructured bricks, resampled into structured bricks, or “empty” bricks that will be rendered as grid boundary only. Direct volume rendering of bricks proceeds in front to back order with one raycasting pass per brick, determining ray start positions based on depth peeling, and also rendering the grid’s boundary geometry in correct visibility order.

- Exact grid boundary visualization
- Scalability in image quality and memory consumption

2 Related Work

The most common object-order methods for rendering unstructured grids are based on the Projected Tetrahedra algorithm [22], which samples tetrahedra only at their bounding faces. Depending on graphics hardware features, the volume rendering integral can be solved using improved accuracy [24], with the best results generally achieved by pre-integration [20]. However, a major performance bottleneck of all PT variants is the need for explicit visibility sorting. A variety of powerful sorting approaches have been developed [34, 24, 23], including hybrid CPU/GPU methods such as HAVS [4]. An explicit sorting step is also required by most point-based methods for rendering unstructured grids [35]. Point-sampling strategies have also been employed successfully for simplification of very large unstructured grids [27]. However, the sorting step required by all object-order methods can only be neglected when commutative blending modes such as purely emissive volumes [30] are used. In contrast, image-order approaches such as raycasting [29, 11] are usually slower in pure rendering performance than cell-projection techniques, but compensate for this fact by the lack of an explicit sorting step. Raycasting approaches are also very flexible, e.g., with respect to adaptive sampling, and are a natural choice when complex non-linear interpolation techniques such as mean value coordinates [13] are desired. For structured grids, GPU raycasting approaches have also been shown to work very well [15]. In order to tackle very large volumes, many approaches employ a hierarchical subdivision, e.g., octrees for a multi-resolution representation and rendering of structured grids [16, 32, 2]. Client/server architectures also are a very powerful approach to rendering very large unstructured grids [3]. Another possibility to render unstructured grids is to resample them into a structured representation for rendering, which can also be performed hierarchically [18]. Powerful resampling algorithms have been developed, especially in order to leverage the rasterization power of GPUs [28, 33]. A recent approach is to use a resampling strategy on-the-fly during rendering [10]. Many cell sorting and also raycasting techniques require non-convex grids to be convexified [14, 19] before they can be rendered. Our approach circumvents this by using a depth peeling approach [7, 1].

The hybrid renderer discussed in this paper has been implemented as a plugin for the SimVis system [5]. Here multiple linked views are used to concurrently show, explore and analyze different aspects of multi-variate data. 3D views of the volume can be used to visualize features which can be specified interactively in several types of attribute views, e.g., scatterplots or histograms. The user chooses to visually represent selected data attributes in such a view, thereby gaining insight into the selected relations within the data. Then, the interesting subsets of the data are brushed directly on the screen. The result of such a brushing operation is reintegrated within the data in the form of a DOI volume. This DOI attribution is used in all views of the analysis setup to visually discriminate the interactively specified features from the rest of the data in a focus+context visualization style which is consistent in all (linked) views. In the SimVis system smooth brushing [6] (enabling fractional DOI-values) as well as the logical combination of brushes for the specification of complex features (based on multiple data attributes and derived information) are supported. As a part of this system the presented raycasting method fully complies to the feature-based focus+context visualization conventions used throughout SimVis.

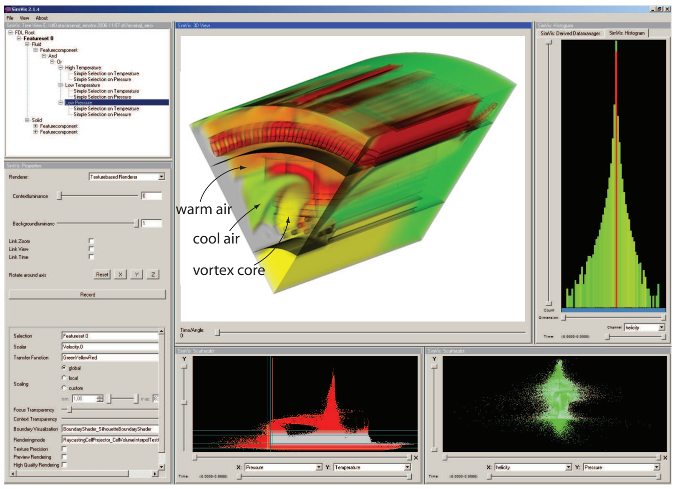

Fig. 2. The SimVis system in conjunction with our raycasting method showing high/low temperature (red/green) regions and low pressure (yellow) regions within the Generator dataset which contains approx. 10^6 cells.

3 Hybrid Raycasting

Figure 1 shows an overview of our hybrid raycasting pipeline. It consists of three major stages. The first stage (Section 3.1) is a pre-processing stage that subdivides the entire unstructured grid into a kD-tree hierarchy by clipping grid cells against kD-tree leaf boundaries, which determines the bricks used in the interactive stages.

The second stage (Section 3.2) must be invoked whenever the degree of interest (DOI) specification or selection of scalar data channel such as “temperature” changes, which is done interactively by the user. According to the DOI, bricks are classified in order to determine their importance and corresponding rendering quality and style. Furthermore, the scalar data in a brick also influence the choice of rendering quality. With regard to quality, the major choice is whether a brick should be rendered in its original unstructured form, or as a lower-quality structured brick. The latter are obtained by on-demand resampling whenever a brick that has not been resampled before is required in structured form. Finally, when the DOI for all values in a brick is zero, it is classified as “empty” brick, which means that only its contained geometric grid boundary is rendered. Note that brick classification is not necessary for every rendered frame, but only when the user actually modifies the DOI or selects a different data channel to display.

The final stage renders all bricks in front-to-back visibility order by rendering the contained mesh boundary and performing direct volume rendering of the mesh interior. This stage is invoked whenever a new view needs to be generated. In order to generate the final image, the kD-tree is traversed in front-to-back visibility order, raycasting each encountered brick in turn and performing compositing in the output image buffer. As illustrated in Figure 3, viewing rays can traverse different brick types, and the sampling pattern and kind of interpolation along each ray is adapted to the current underlying brick type. Especially due to the hybrid nature of our pipeline, the transition between individual bricks must be handled with care (Section 3.3). Raycasting of individual bricks depends on the underlying type, such as unstructured bricks (Section 3.4) or structured bricks (Section 3.5). For each brick, the front-most mesh surface in the brick is rasterized first, in order to obtain ray start positions. The mesh surface itself is rendered as well, and direct volume rendering with one of several user-selectable optical models is performed in the interior of the mesh between any two successive mesh boundary intersections. This ensures that the original mesh boundary is rendered accurately and consistently everywhere and independent of actual brick types. Non-convex parts of the mesh are handled using a depth peeling approach. For empty bricks, the mesh boundary is rendered using regular depth peeling [7]. For unstructured bricks, the raycaster is started multiple times in order to cast each depth layer individually and perform proper compositing [31].

3.1 Spatial Subdivision

Using a spatial subdivision scheme to cope with large datasets is a common approach, when dealing with structured data volumes. But until now subdivision has only been used on unstructured data which has been resampled to a structured grid [18]. The hybrid raycasting method proposed in this paper is based on a kD-tree decomposition of the original unstructured data volume into bricks, which allows for several optimizations. We decided to use a kD-tree, since it provides better control over the number of entries in its child nodes than comparable subdivision schemes like Octrees. It is important to note, that every brick's contents have to be properly clipped to its boundaries in order to guarantee correct composition of the individual brick renderings. Depending on the bricks representation three different clipping methods are used: A CPU based cell intersection algorithm yielding the intersection polygon for structured bricks, a GPU based intersection approach based on clipping a quad to a cell's interior for unstructured bricks and simple clipping planes for the boundary of empty bricks. The depth of the kD-tree is determined by limiting the number of cells and vertices within a brick to 64K, which allows for memory savings when storing unstructured grid topology. If time-varying meshes, as described in Section 5.2, have to be visualized the brick decomposition is performed for every timestep (this is not necessary for time-dependent data specified on static grids).

An important advantage of the spatial subdivision is that we are now able to apply straight forward optimizations commonly used by brick based structured volume visualization approaches to unstructured data as well. We perform view frustum culling on the individual bricks, and empty bricks are skipped entirely (by the volume renderer), helping us to distribute resources only among visible portions of the data. Besides these obvious advantages of using a spatial subdivision, it also helps us dealing with highly non convex meshes more efficiently. Many datasets from the engineering field have very complex surface geometries (see Section 5.1), which would require a high number of rendering passes, when treated as one entity. But with the introduction of a spatial subdivision scheme the number of depth peels, which have to be used to produce a correct visualization can be reduced greatly because of the implicit visibility order of the bricks within the kD-tree.

Celltype	Faces	Vertices	Min. No. of Tetrahedra
Tetrahedron	4	4	1
Pyramid	5	6	2
Prism	5	6	3
Octahedron	8	6	4
Hexahedron	6	8	5

Table 1. Cell types supported by the SimVis system.

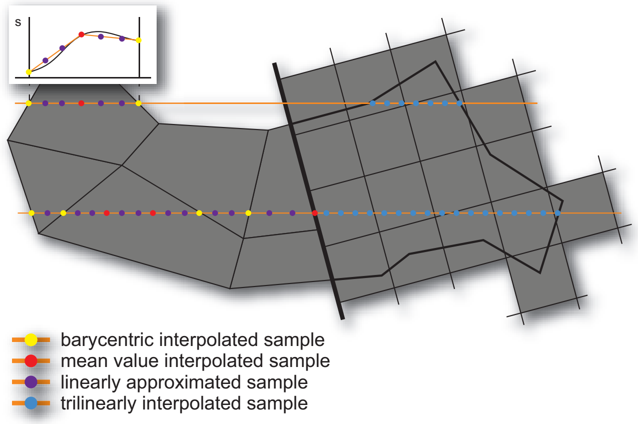
s
barycentric interpolated sample
mean value interpolated sample
linearly approximated sample
trilinearly interpolated sample

Fig. 3. This illustration shows two different view rays through an unstructured (left) and structured (right) brick. The different sampling types are represented by the colored dots along the rays.

3.2 Brick Classification

Since every brick of the data volume is treated separately by the renderer it is possible to distribute processing and memory resources better, favoring important portions of a dataset over less interesting regions. In order to determine the importance of a brick's contents we propose to combine three different simple measures, which can be evaluated efficiently. The first measure is the average DOI value within a brick, which represents the interest, a user has specified for its contents. Additionally the entropy of the histograms of the DOI and the scalar fields within a brick are added to the measure in order to reflect its information content. All three measures result in values in [0, 1] and are weighted equally. Now the individual bricks can be sorted based on their importance and texture memory resources can be distributed based on this order. The original unstructured grid data of a brick is used as long as enough memory is available. If bricks exist, whose contents wont fit into the portion of texture memory allocated for unstructured data, a small 3D-texture is allocated and resampling is performed. Since empty bricks do not need any volumetric representation they are ignored by the memory management system.

3.3 Inter-Brick Ray Propagation

An important issue in bricked volume rendering is that the transition between individual bricks does not introduce visual artifacts. Naturally, this issue is especially important in a hybrid bricking scheme such as the one presented here. Specifics of raycasting unstructured and structured grids, respectively, are described in the two following sections, whereas this section highlights common details and describes how rays are propagated from brick to brick.

Each ray cast through a brick starts either at the front-most mesh boundary contained in the brick, or at the intersection of the brick boundary with the interior of an unstructured cell. This is true for both unstructured and structured bricks, and ray start positions are obtained via rasterization of either mesh boundary faces or cell/brick-boundary intersection geometry. Compositing is performed in front-to-back order using two compositing buffers that are used alternately (*ping-pong blending*).

Figure 3 shows the two most important cases for ray traversal from one brick to the next. In a single rendering pass, each ray stops either when the mesh interior is exited, which may happen multiple times in a single brick due to non-convex mesh geometry and depth peeling, or when a back-face of the brick bounding box is hit. Depending on these cases, raycasting in the next brick continues either at the exact same location where the previous ray segment has stopped (lower ray in Figure 3), or continues where the ray re-enters the mesh interior (upper ray in Figure 3). This property is consistent between brick transitions of the same type (e.g., unstructured to unstructured) and mixed type (shown in Figure 3).

Within a brick, non-convex mesh geometry is handled via depth peeling, and thus raycasting may be performed in multiple rendering passes. In principle, each subsequent pass continues at the next ray entry position behind the depth, where the previous pass has stopped, until the last depth layer has been traversed. The number of required rendering passes is determined via OpenGL occlusion queries, which determine whether rendering an additional depth layer is necessary. Note that the maximum number of depth layers within a single brick is much lower than for the entire grid. Thus, bricking significantly reduces the number of rendering passes due to depth peeling and thus alleviates its potential performance impact.

3.4 Unstructured Bricks

Our raycasting algorithm for unstructured bricks is based on the work of Garrity [9] and Weiler et al. [29, 31] for tetrahedral meshes. However, we perform direct raycasting of non-tetrahedral cells without performing a (often ambiguous) tetrahedral decomposition, which usually introduces C^1 discontinuities of the interpolated scalar within a cell. Figures 4 (a) and (b) show volume renderings of two different tetrahedral subdivisions of an octahedron, with color representing the influence of the top-most vertex. In contrast, Figure 4 (c) uses our algorithm without decomposition, and interpolation using mean value coordinates.

Data and Memory Management: Raycasting of one brick is performed for small submeshes of the whole dataset, which have to be represented efficiently in texture memory. We employ a custom memory manager to distribute texture resources between bricks. Figure 5 shows the three different types of *texture sets* we are using to store the data of unstructured bricks: the *cell texture set*, *cell face texture set*, and *cell vertex texture set*. The cell texture set stores the number of faces of each cell. For cell-centered data, it also stores the per-cell DOI and the data scalar. The cell face texture set stores the neighboring cells' indexes, face plane equations, and vertex indexes. The cell vertex texture set stores vertex (x,y,z) position, and per-vertex DOI and data scalar. Each texture set is comprised of one or more 2D textures of size $256 \times r_{max}$, where r_{max} is the maximum allowed size of a 2D texture (or less). Using textures of width 256 allows to quickly obtain 2D coordinates from a 1D index by using the lower 8 bits directly for indexing within a row, and the higher bits for addressing the row relative to a row base index known for each brick. For each brick as many consecutive rows as necessary are allocated, forcing an integral number of rows to make address computations simpler. I.e., the number of allocated rows in the cell texture set for a brick containing n cells is $\lceil n/256 \rceil$, where in our implementation $n = 64K$.

In order to avoid storing an additional index that points from cell data to face data, we use a simple trade-off: In each brick we know the maximum number of faces per cell, and allocate this maximum number of entries for all cells in the brick. As shown in Table 3 this is feasible since cells with a high number of faces are more commonly used within unstructured meshes in order to reduce the number of volume elements which have to be simulated. Thus, $\lceil (n \cdot f_{max})/256 \rceil$ rows have to be allocated in the cell face texture set for a brick with cells containing a maximum number of f_{max} faces. Indexing within this data structure is straight-forward: a linear transformation of the index used for the cell texture set suffices. Each face data item stores four vertex indexes that can be used to look up the corresponding vertex data in the cell vertex texture set. Overall, this data organization scheme is designed for fast and simple access to the data and topology of the underlying mesh. At most one indirection has to be used to retrieve the complete data for one cell.

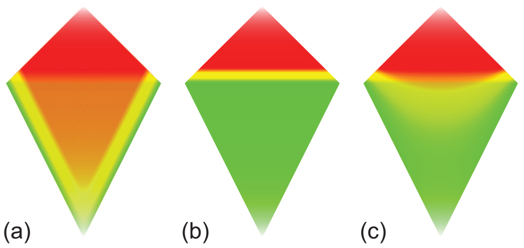

Fig. 4. Comparison of the influence of the top-most vertex of an octahedron in two different tetrahedralizations (a) and (b), and the original cell using mean value interpolation (c).

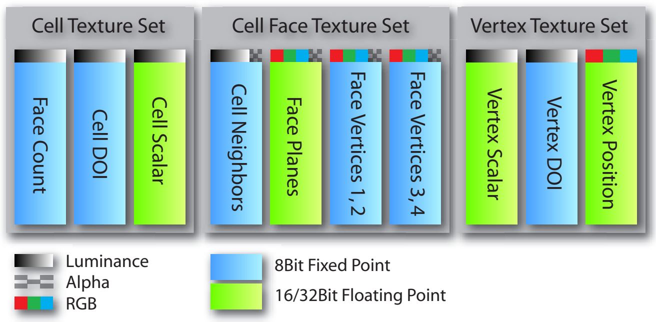

Fig. 5. Different *texture sets* and the textures contained within them.

Raycasting: The raycasting process itself consists of two major parts: ray propagation through the grid, and sampling the scalar function within a cell in order to evaluate the volume rendering integral. Rays are started by rasterizing the surface of the mesh in order to write encoded cell indices into the red and green color channels. Since we limit the maximum number of cells per brick to 64K, two channels with 8 bits each are sufficient. Additionally, the face through which a ray enters a cell is encoded in the blue channel. If a cell is clipped by the brick boundary, this is indicated in the blue channel as well. During ray propagation, only cell-centered and cell face information is used, which can be directly addressed allowing for rapid evaluation of the face through which a ray exits the current cell. This is achieved by intersecting the ray with all face planes and choosing the intersection closest to the entry point in the ray's direction [9].

Sampling and Interpolation: The volume rendering integral is approximated by sampling the DOI and scalar data volumes within each cell and compositing the corresponding opacities and colors according to one of several optical models (Section 4). In contrast with approaches for tetrahedral cells, sampling a single cell multiple times is necessary in order to handle complex cell types (Table 1). In order to obtain scalar values at arbitrary positions within a cell, we employ mean value coordinates for interpolation as introduced by Floater [8], and generalized to triangular meshes by Ju et al. [13]. The following important properties make this kind of interpolant ideal for our purposes:

1. Convergence toward 2D barycentric interpolation on cell faces.
2. Proper reconstruction of linear functions sampled at cell vertices.
3. Efficient evaluation without much cell topology information.

The first property guarantees that the interpolated function

$$\hat{f}(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x}) f_i}{\sum_i w_i(\mathbf{x})}, \qquad (1)$$

with scalar values f_i at the cell vertices, and interpolation weights w_i, is C^0-continuous across cell boundaries. Close to a cell face, only the vertices of this face contribute to the interpolation. The second property implies that the linearity of 3D barycentric interpolation of a tetrahedral decomposition of a more complex cell is not lost. For a linear function $f_l(\mathbf{x})$ sampled at the vertices, the interpolated function $\hat{f}_l(\mathbf{x}) = f_l(\mathbf{x})$. Since the weights $w_i(\mathbf{x})$ have to be evaluated efficiently in order to allow for interactive frame rates, and texture memory is limited as well, the third property is also of very high importance. All vertex weights can be computed by treating each cell face separately without considering adjacent faces. Each vertex weight w_i can be split up into components $\tilde{w}_{j_k}$ that are contributed by the faces j that contain the vertex at index k. Now $\tilde{w}_{j_k}$ can be evaluated for position $\mathbf{x}$ as

$$\tilde{w}_{j_k}(\mathbf{x}) = \frac{\mathbf{n}_{j_k} \cdot \mathbf{m}_j}{\mathbf{n}_{j_k} \cdot (\mathbf{p}_{j_k} - \mathbf{x})}. \tag{2}$$

Here, $\mathbf{p}_{j_k}$ is the position of vertex k in face j, and $\mathbf{n}_{j_k}$ the normal of the plane defined by the position $\mathbf{x}$ and vertices $k+1$ and $k+2$ of face j. The vector $\mathbf{m}_j$ is the so called mean vector of face j which can be computed as

$$\mathbf{m}_j = \sum_{k=1}^{3} \theta_{j_k} \mathbf{n}_{j_k}. \tag{3}$$

Figure 6 illustrates the nomenclature used in greater detail. For a more elaborate description of mean value coordinates see the original publication by Ju et al. [13], who additionally propose several optimizations to enhance the numerical robustness of their approach in the vicinity of the polyhedron's faces and vertices.

Evaluating all contributions of the vertices of all cell faces to one sample position means iterating over all faces, which implies that their vertex data either have to be stored temporarily while sampling a cell, or have to be read from the corresponding textures for each sample. In order to avoid re-computations, as well as too many temporaries, we use the following optimization: Instead of computing all samples one by one, the contribution of each face to all samples is computed at once. Thus, only the data corresponding to one face and three scalar values per sample have to be stored temporarily while sampling a cell: the current scalar value, DOI value, and the accumulated weight which is used to homogenize the samples after all faces of a cell have made their contributions.

Sample Distribution: Since the computation of mean value coordinates for a sample is expensive (it involves several cross products, normalizations, and an inverse trigonometric function to compute the arc length θ_{j_k}), it is desirable to distribute mean value samples carefully. Thus, mean value interpolation is only used in a cell's interior, distributing the samples equally along the ray segment within it. Simple barycentric interpolation is applied to compute scalar values at the ray-face intersections. In order to further improve the image quality without sacrificing much performance, additional samples can be computed between the positions of the already computed samples by linearly interpolating between them. If only one scalar volume is used, pre-integration methods from tetrahedral grid raycasting could be applied [20] instead of computing multiple linearly interpolated samples.

In order to steer the performance/image quality of the visualization, two parameters are used by the raycaster for unstructured bricks: The maximum length of a ray segment which can be approximated

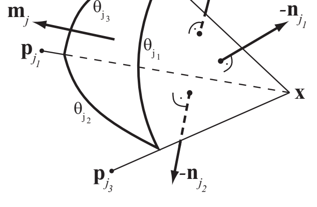

Fig. 6. Mean value coordinates are based on the projection of all faces of a polyhedron onto a unit sphere centered at the position for which the coordinates are computed [13].

linearly (l_l), and the maximum length of a ray segment along which a scalar is considered to be constant (l_c). Thus $\lfloor s_l / l_l \rfloor$ mean value samples are computed for a ray segment within a cell, where s_l is the length of the ray segment. We have chosen to distribute the sampling positions evenly to avoid mean value coordinate computations near the face through which the view ray exits, which degenerates towards barycentric interpolation anyway. The parameter l_c is used to determine how many samples should be taken in-between two mean value/barycentrically interpolated values by interpolating linearly, as shown in Figure 3. Again $\lfloor s_c / l_c \rfloor$ equally-distributed samples are computed and evaluated, where s_c is:

$$s_c = \frac{s_l}{\lfloor s_l / l_l \rfloor + 1}. \tag{4}$$

Both parameters can be used to specify several special cases that can either be used to render preview images or to produce high quality presentation visualizations. By setting s_l larger than the diameter of the largest cell's bounding sphere, only barycentric samples are computed at the cell's faces. This reduces rendering complexity tremendously and results in highly interactive framerates even for large datasets. In order to create very high quality images, s_l can be specified far smaller than the largest cell, and s_c larger than s_l, which forces the raycaster to perform only mean value interpolation.

3.5 Structured Bricks

If the brick classification step (Section 3.2) determines that a brick should be rendered in structured form, all unstructured cells contained in or intersected by the brick will be resampled into a single 3D texture. The resampling resolution depends on the number of unstructured cells in the brick instead of its volume. For this purpose, a uniform distribution within the brick is assumed in order to guarantee memory savings over the unstructured representation. For n cells in the brick, we use a power-of-two resolution that is not smaller than $\sqrt[3]{n}$ for the longest brick axis.

In this work, we are not focusing on the actual algorithm used for resampling, as any existing method could be used for this purpose, e.g. [33]. Currently, we are using a very simple and fast CPU-based resampling algorithm that still achieves good results. Basically, the scalar values at the resampling locations are determined by Gaussian splats positioned at the cell centers. The size of these splats is chosen according to the cell size, and modified if necessary such that even cells that are much smaller than a single voxel still contribute to the eight surrounding voxels.

Raycasting of the resulting structured bricks is performed by rasterizing the mesh boundary in order to obtain ray start and end positions respectively. In order to handle non-convex grids and skip all empty space, we use a modified depth peeling approach [31], where the number of raycasting passes is determined by the number of depth layers. Depth peeling also makes rendering the geometric grid boundary in correct visibility order straight-forward. The main difference to regular volume raycasting [15] is that rays are started and stopped, respectively, at exactly those positions where a ray enters and exits the unstructured mesh, respectively. The mesh boundary is rendered into two floating point textures: The first one for the position of the ray entry point, and the second one for the ray exit point. Depth peeling is also performed separately for ray entry and ray exit positions, respectively. The main reason for this is that common simulation approaches such as coupled heat transport and flow simulation result in meshes where interface faces are duplicated, i.e., both the front-face and the back-face are present in the mesh. Section 5 describes an example. The sampling rate for structured bricks is chosen to be consistent with unstructured brick rendering. The parameter l_c described in the previous section that determines the maximum distance between two linearly interpolated samples in unstructured bricks also determines the sampling rate of structured bricks. As shown in Figure 3 and described in Section 3.3, the actual positions of samples are chosen such that the visibility of the brick boundary between different brick types is minimized.

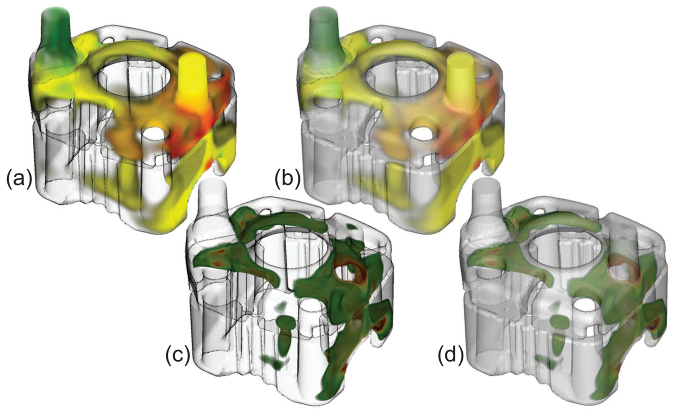

Fig. 7. Two different boundary rendering techniques (surface and silhouette enhancement) combined with two different optical volume models (standard DVR and smooth iso-surfaces) to four different rendering modes. High turbulent kinetic energy (TKE) is brushed and pressure mapped to color in the DVR images and used as parameter for the iso-surface visualizations.

4 Rendering Modes

The hybrid raycasting method proposed in this paper has to deal with two different data volumes simultaneously: the scalar field of the underlying dataset (e.g., temperature), and the scalar DOI function specified by the user. In order to visualize this multi-volume, several rendering modes can be selected by specifying a boundary visualization technique, and an optical model for direct volume rendering. Figure 7 illustrates all four combinations of two different optical models and boundary visualization techniques. Table 2 summarizes the major parameters and the underlying data required. The most common rendering mode uses an emission/absorption optical model without gradient-based shading. In order to compute a color $c(\mathbf{x})$ at position $\mathbf{x}$ based on the scalar data volume in combination with the DOI function, the following transfer function is used:

$$c(\mathbf{x}) = DOI(\mathbf{x}) \cdot tf_w\big(s(\mathbf{x})\big) + \big(1 - DOI(\mathbf{x})\big) \cdot c_l. \tag{5}$$

The opacity is linearly derived from the DOI function:

$$\alpha(\mathbf{x}) = DOI(\mathbf{x}) \cdot f_\alpha. \tag{6}$$

Figure 7 (a) and (b) show this optical model applied to a small cooling jacket dataset in which regions of high turbulent kinetic energy have been selected smoothly. Color values represent the corresponding pressure within the volume. The context luminance c_l is set to zero to emphasize the structure of the selected data region.

The second optical model shown in Figures 7 (c) and (d) creates visualizations that resemble smooth iso-surfaces. The windowed scalar data interval from tf_{min} to tf_{max} is used to determine the opacity of a sample through the following equation:

$$\lambda(\mathbf{x}) \quad = \quad DOI(\mathbf{x}) \cdot t_{01}\big(s_{cl}(\mathbf{x})\big) \tag{7}$$

Rendering Mode Parameters		**Available Data**	
f_α	global opacity of focus	$DOI(\mathbf{x})$	DOI function
c_α	global opacity of context	$s(\mathbf{x})$	scalar field
c_l	global luminance of context	$\mathbf{v}$	view direction
b_c	boundary color	$\mathbf{n}_s$	surface normal
$tf(s)$	user-defined transfer function		
tf_{min}	lower windowing bound		
tf_{max}	upper windowing bound		
$tf_w(s)$	$= tf(\frac{s - tf_{min}}{tf_{max} - tf_{min}})$; windowed tf		

Table 2. User-definable parameters, and the data available to the different boundary and volume rendering algorithms.

$$\alpha(\mathbf{x}) \quad = \quad \lambda(\mathbf{x}) \cdot f_\alpha, \tag{8}$$

where $t_{01}(\cdot)$ is a tent function centered at 0.5 in the interval $[0,1]$, and $s_{cl}(\mathbf{x})$ is the scalar value $s(\mathbf{x})$ mapped and clamped from the interval $[tf_{min}, tf_{max}]$ to [0,1]. These equations have to incorporate the DOI function in order to ensure that bricks with $DOI(\mathbf{x}) = 0$ for all $\mathbf{x}$ also use a constant opacity of zero. For such bricks only the boundary will be rendered. Thus, the smooth iso-surface created by this optical model can additionally be clipped smoothly by specifying an adequate DOI function. The equation for assigning color values to a sample is similar to equation 5:

$$c(\mathbf{x}) = \lambda(\mathbf{x}) \cdot tf_w\big(s(\mathbf{x})\big) + \big(1 - \lambda(\mathbf{x})\big) \cdot c_l. \tag{9}$$

By using $\lambda(\mathbf{x})$ to modulate the resulting color the lack of proper shading of the smooth surface can be somewhat compensated. Rays only touching the surface will gather more dark samples (if c_l is set to zero) than rays intersecting it, which results in dark silhouettes. This "shading" approach is similar to limb darkening [12].

It is important to note that besides the optical model for volume rendering the boundary visualization is of very high importance to the overall visualization approach presented in this paper. Datasets from the engineering field very often contain complex surfaces which can be of high interest to the user if they interact with the simulated phenomena. Therefore, we have decided to treat the mesh surface as context information which can be visualized using different boundary visualization techniques. Figures 7 (a) and (c) show silhouette boundary visualizations, whereas Figures 7 (b) and (d) have been created using simple shaded transparent surface rendering.

Silhouette rendering is realized by directly assigning b_c to the color of the boundary sample and computing the corresponding opacity as follows:

$$\alpha = (\mathbf{n}_s \cdot \mathbf{v})^4 \cdot c_\alpha. \tag{10}$$

With this approach only a minimal amount of the data volume is obstructed by the boundary visualization, which, however, still provides proper contextual cues to relate the selected flow features to the overall geometry of the dataset.

If the surface of a dataset is extremely complex and the silhouette visualization becomes too cluttered, semi-transparent surface rendering can be used. Then, a simple diffuse lighting equation is evaluated based on the surface normal and directional light sources in and against the viewing direction. The c_α parameter is directly used as opacity. Even though this boundary visualization technique obstructs the data volume stronger than the silhouettes, the visualization result is less cluttered. This allows the user to make a trade-off between a clear visualization of flow features with minimal geometrical context, or a less clear feature visualization that is embedded better in the surrounding mesh. Which approach should be favored highly depends on the inspected dataset and the features of interest.

5 Application Results

In this section some example applications of the hybrid raycasting algorithm in conjunction with the SimVis system are presented. Two very different datasets are used to demonstrate the power of our rendering approach. Furthermore some image quality and performance considerations are presented.

Due to space limitations some of the presented images for this application section are rather small, higher quality images as well as some further results and the accompanying video are available from the project homepage at `www.vrvis.at/via/research/hybridRC`.

5.1 Large Data

The generator dataset is courtesy of Arsenal Research, Vienna Austria and Traktionssystem Austria. It contains one sixth of the geometry of a generator (the remaining five sixth are symmetrical). The data volume itself consists of two parts: the solid portion of the generator and the air surrounding it. Both parts are separate meshes which touch each other on a common surface. The problem, which has been modeled

Dataset	Tet.	Pyra.	Prism.	Hexa.	Overall	Tetrahedralized	Bricks	Interact.	Static.	Presentation
Generator	55.4%	0.6%	23.9%	20.1%	6,729,806	15,405,842	284	940ms	2969ms	22s
Large Cooling Jacket	0.2%	5.2%	9.3%	85.3%	1,537,898	7,149,388	57	262ms	1109ms	2430ms
Two Stroke Engine	0.8%	4.3%	8.4%	86.5%	149,864	700,091	8	50ms	153ms	153ms
Small Cooling Jacket	0.2%	1.3%	2.1%	98.4%	76,816	377,170	2	53ms	784ms	784ms

Table 3. Various datasets used throughout this paper.

within the dataset, is the cooling of the generator. Thus heat transport and dissipation has been simulated for the solid portion of the dataset, while air temperature and flow has been computed for the surrounding air.

The grid itself poses multiple challenges to the visualization system. First of all the mesh surface has a very high depth complexity, which means, that many depth layers have to be rendered to produce the final visualization. Even when decomposed into 284 bricks every brick contains between 2 to 8 depth layers (depending on the view port). On average 5 depth layers have to be rendered. Since nearly all state of the art methods can only handle tetrahedral volume cells the generator dataset would have to be tetrahedralized to be visualized with them, which would result in a cellcount of over 15 million as shown in Table 3.

The setup within the dataset assumes, that air is being sucked through the generator from one end to the other, while the central part of it, which is called the rotor, is rotating. This rotation induces electric current in the surrounding coils, which are thus heated up. In order to increase the efficiency of the generator it is necessary to provide efficient cooling, which means, that the air flow around the winding heads surrounding the rotor has to be optimized. Figure 8 (a) gives a rough overview over the dataset geometry by showing only the solid portions of the generator with color representing the temperature. As expected the winding heads at the air inlet are cooler than at the air outlet.

Figure 2 shows a result image from a complete visualization session using the SimVis system in conjunction with the hybrid raycasting algorithm. It is comprised of four different selections which are combined by using a fuzzy or operation and maps temperature to color. The first selection specifies the solid portions of the dataset which are used to provide additional context to the user. Additionally cold portions of air have been selected. Here it is notable that a great amount

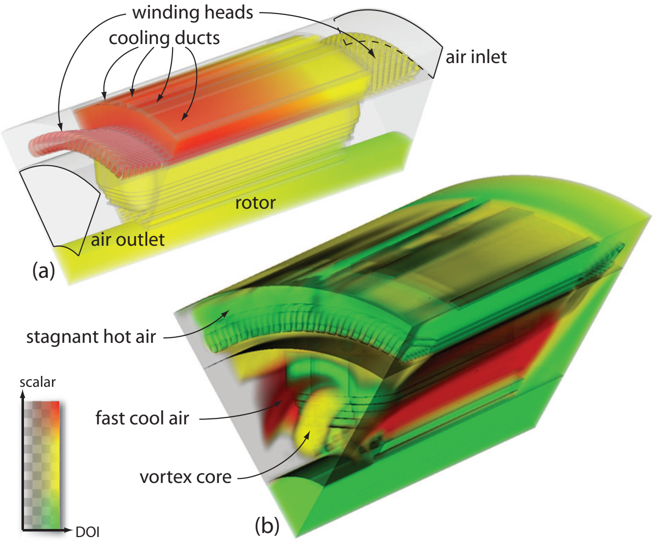

Fig. 8. (a) An overview over the Generator dataset depicting the different parts of the simulated volume. Standard DVR and the surface rendering technique have been combined to create this image. (b) The same selections as shown in Figure 2 but velocity magnitude mapped to color.

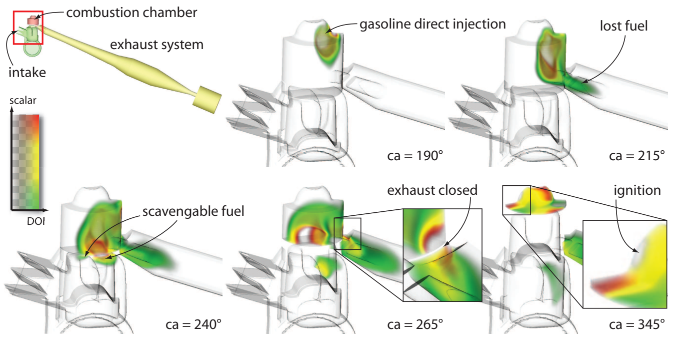

Fig. 9. Two Stroke Engine dataset shown at different timesteps which are denoted by the current crank angle (ca).

of relatively cool air moves (very rapidly as depicted in Figure 8 (b)) through the openings left and right of the rotor. These air masses mainly contribute to the cooling of the winding heads at the air inlet and are sucked into a vortex at the air outlet. The third selection consists of warm portions It can be observed that cool air which is sucked through the cooling ducts at the top of the dataset is heated up rather quickly and stagnates in the space above the winding heads, which can be confirmed in Figure 8 (b), which shows the same selection as Figure 2, but with flow velocity mapped to color (the green region above the winding heads represents hot and slow air). In order to visualize the vortex, which is located behind the air outlet the fourth selection marks low pressure as important. As concluded by Trenker et al. [26] (who used the SimVis system in conjunction with an early prototype implementation of the hybrid raycasting method) it is desirable to deflect this vortex to pass through the winding heads at the air outlet (probably by moving the outlet itself) to avoid the stagnation of hot air and thus provide better cooling for the overall machine.

5.2 Time-Dependent Data

The Two Stroke Engine dataset is courtesy of the Institute for Internal Combustion Engines and Thermodynamics, Technical University Graz, Austria. It contains the simulation of a loop scavanged two stroke engine for which the fuel injection and combustion process has been modeled. An in depth analysis of this dataset has been performed by Schmidt et al. [21] who compare two different fuel injection pressures (in the simulated engine gasoline direct injection is used) with respect to the fuel distribution within the combustion chamber at the ignition time and the loss of combustable mixture through the exhaust. Since the flow in the intake (through which fresh air is sucked into the combustion chamber, if the intake ducts are not blocked by the piston) and exhaust ducts is essential to the functioning of the engine those parts have been modeled additionally to the combustion chamber as shown in Figure 9 (top left).

The machinery modeled in the Two Stroke Engine dataset is not static (the piston is moving up and down) which means that the underlying volume mesh changes over the simulated time span: vertex positions are modified and the overall mesh topology changes in order to account for the degeneration of cells. The fuel injection itself starts at 165° crank angle (ca) whereas the ignition is performed at 345° ca. The scalar values of highest importance in this dataset are the equivalence ratio, which is the ratio between fuel and air and the reaction progress variable, which represents the progress of the combustion (zero representing unburnt and one fully burnt mixture). Thus

Figure 9 shows a smooth selection of optimal equivalence ratio (from 0.7 to 1.4) which is additionally restricted by marking only portions of the data, where no burning has occurred (reaction progress variable equaling zero).

In Figure 9 different timesteps, denoted by the current crank angle (ca), of these selections are shown. At ca = 190° the gasoline injection is nearly completed and the air intake ducts and exhaust system opening are being closed by the piston. Successively some of the gasoline starts leaking through the exhaust as shown at ca = 215°. Here only portions of a lean mixture are lost which is indicated by the green color. At ca = 240° some gasoline is being sucked into the intake system, which will be scavenged in the next combustion cycle. Additionally portions of very rich mixture (indicated by red) move down towards the exhaust opening and are sucked out just before it is closed at ca = 265°. The final image taken at ca = 345° depicts the gasoline distribution shortly after the ignition, showing that the ignition spark itself is located well within a region of good equivalence ratio (indicated by yellow). Additionally it can be noted that the overall distribution of burnable mixture within the combustion chamber is highly uneven leading to higher emissions and lower efficiency in this case (Schmidt et al. [21] have shown that higher injection pressure can lead to a more homogenous gasoline distribution within the combustion chamber). It is obvious that especially for simulations like this, the proper visualization of mesh boundaries is of very high importance since they indicate the moving parts of machinery which strongly influence complex flow conditions within the simulated domain.

5.3 Quality and Performance Considerations

There is a huge set of different parameters which can be used to steer the memory consumption, the speed and the image quality of the presented visualization approach. In order to guarantee responsiveness during an analysis and exploration session two different sets of parameter settings can be chosen: one being applied during interaction and one if a static image should be generated. Additionally it is possible to progressively update the display during the generation of a static image (every n'th brick the accumulated color and opacity information is copied to the front buffer). In Table 3 some performance measures are presented for the different datasets shown in this paper. Different quality settings have been used to measure rendering performance during interaction, for a static image and for the dataset without using resampling (here a brick always forces the memory manager to allocate texture memory, even if data from another brick has to be overwritten). During interaction only one depth peel for every brick is rendered, and the unstructured brick sampling settings are configured to perform only barycentric interpolation at the cell boundaries. Additionally the result image resolution is reduced to 256^2 (instead of 512^2 for the static measurements). In the static case the sampling settings for the unstructured bricks are set to $l_l = c_{max}/4$ and $l_c = c_{max}/16$ with

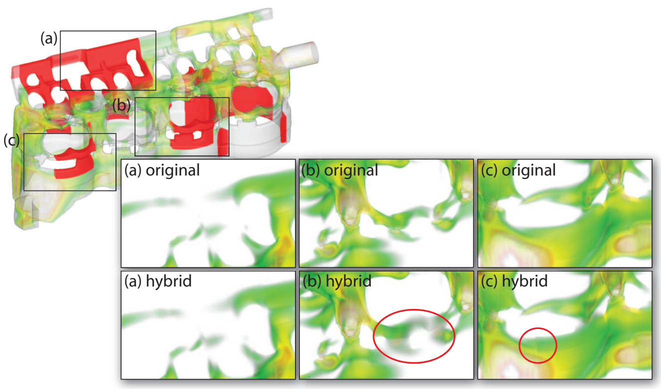

Fig. 10. Overview and magnifications of the Large Cooling Jacket dataset comparing the low quality resampled to the original unstructured representation. Resampled bricks are shown in red in the overview.

c_{max} being the maximum diameter of a cell's bounding sphere. Thus a maximum of four mean value samples are computed per cell per ray. These settings are equal to those used to create the "Presentation" column of Table 3 resulting in the same measures for datasets which completely fit into the texture memory allocated for unstructured data. All test have been carried out on an AthlonX2 4400+ with 4GB of ram and a Geforce 8800GTX with 768MB video memory.

Besides the rendering performance itself the brick boundaries are of high importance when image quality has to be considered (especially between structured and unstructured bricks). Figure 10 shows some comparisons between a high quality visualization using only unstructured bricks and the default still image settings mentioned above (in order to reduce obstructions the boundary visualization is set to completely transparent). The overview in the top/left corner indicates resampled bricks in red. In Figure 10 (a) barely any difference between resampled and original mesh can be seen, whereas Figure 10 (b) shows the low pass filtering effect of the resampling process. Additionally Figure 10 (c) shows a slight discontinuity between the resampled and original mesh. It should be noted that the previously mentioned effects mostly occur if high frequency transfer functions and DOI specifications are used. But since the SimVis framework emphasizes the smoothness of flow features (through smooth brushing) and the resampling process can be steered by interactively modifying the DOI specification the application of resampling to unimportant regions is feasible.

6 Conclusions and Future Work

We have presented a scalable hybrid GPU raycasting algorithm for unstructured grids. Our method directly renders complex cell types without tetrahedralization, where non-tetrahedral cells employ mean value interpolation for chosen samples and interpolate linearly in-between. We employ bricking, resampling, and custom texture memory management in order to sustain interactive performance and make optimal use of the available amount of texture memory. It is a hybrid approach in the sense that it combines unstructured and structured grid raycasting, as well as image space methods (raycasting) and object space approaches (bricking). Different volume rendering styles are combined with surface rendering methods to create highly parameterizable visualizations that are based on two concurrent data volumes: a degree of interest (DOI) function specified in the SimVis system, and a scalar data volume. We also pay special attention to accurately rendering the surface mesh of the original grid at all times. Apart from better interpolation quality, avoiding tetrahedralization also balances the additional amount of work required by mean value interpolation as well as the additional memory for storing a variety of cell types. Converted to tetrahedral meshes, our datasets would contain twice to four times as many cells. The effectiveness and scalability of our approach has been demonstrated by applying it to two very different real-world datasets: the first one contains a large and highly complex mesh, and the other one is time-dependent and emphasizes its geometric boundary.

There are several interesting areas for future work. The interpolation scheme could be extended to completely arbitrary polyhedra, which are becoming more common in real-world simulation grids. Moreover, new surface and volume rendering models could be integrated easily. The current resampling process could be greatly improved by implementing GPU-based methods. Finally, resampled bricks do not necessarily have to be seen as low-quality substitute of unstructured bricks. If adequate pre-filtering is performed during resampling, rendering those bricks where a single image pixel covers many cells in structured form could provide a way for performing anti-aliasing for high-quality rendering.

7 Acknowledgements

We thank Hermann Lang, Markus Trenker, Stephan Schmidt and Oliver Schögl for their support during this work. The datasets are courtesy of the Institute for Internal Combustion Engines and Thermodynamics, Technical University Graz, Austria, AVL List GmbH, Graz, Austria, Arsenal Research, Vienna, Austria and Traktionssysteme Austria (contact: DDr. Neudorfer).

This work has been partly funded by the Bridge funding program of the Austrian Funding Agency (FFG) in the scope of the Severe Weather Explorer project (Nr. 812122).

REFERENCES

[1] F. F. Bernardon, J. L. D. Comba, C. A. Pagot, and C. T. Silva. Gpu-based tiled ray casting using depth peeling. *Journal of Graphics Tools*, 11.3:23–29, 2006.

[2] I. Boada, I. Navazo, and R. Scopigno. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17(3):185–197, 2001.

[3] S. P. Callahan, L. Bavoil, V. Pascucci, and C. T. Silva. Progressive volume rendering of large unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1307–1314, 2006.

[4] S. P. Callahan, M. Ikits, J. L. D. Comba, and C. T. Silva. Hardware-assisted visibility sorting for unstructured volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):285–295, 2005.

[5] H. Doleisch, M. Gasser, and H. Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proc. of the 5th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2003)*, pages 239–248, 2003.

[6] H. Doleisch and H. Hauser. Smooth brushing for focus+context visualization of simulation data in 3D. In *WSCG 2002 Conference Proceedings*, pages 147–154, 2002.

[7] C. Everitt. Interactive order-independent transparency, Sept. 01 2001.

[8] M. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1), 2003.

[9] M. P. Garrity. Raytracing irregular volume data. *ACM Computer Graphics*, 24(5):35–40, 1990.

[10] J. Georgii and R. Westermann. A generic and scalable pipeline for GPU tetrahedral grid rendering. *IEEE Trans. Vis. Comput. Graph*, 12(5):1345–1352, 2006.

[11] M. Hadwiger, C. Sigg, H. Scharsach, K. Buhler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[12] A. Helgeland and O. Andreassen. Visualization of vector fields using seed LIC and volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):673–682, 2004.

[13] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *Proceedings of SIGGRAPH 2005*, pages 561–566, 2005.

[14] M. Kraus and T. Ertl. Cell-projection of cyclic meshes. In *Proceedings IEEE Visualization 2001*, pages 215–222, 2001.

[15] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings IEEE Visualization 2003*, pages 287–292, 2003.

[16] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings IEEE Visualization '99*, pages 355–361, 1999.

[17] T. Langer, A. Belyaev, and H.-P. Seidel. Spherical barycentric coordinates. In *Proceedings of Eurographics Symposium on Geometry Processing 2006*, pages 81–88, 2005.

[18] J. Leven, J.Corso, J. Cohen, and S. Kumar. Interactive visualization of unstructured grids using hierarchical 3D textures. In *Proc. IEEEE Symposium on Volume Visualization 2002 (VolVis 2002)*, pages 37–44, Oct. 28–29 2002.

[19] S. Röttger, S. Guthe, A. Schieber, and T. Ertl. Convexification of unstructured grids. In *Proc. of the 9th Fall Workshop on Vision, Modeling and Visualization (VMV 2004)*, pages 283–292, 2004.

[20] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *IEEE Visualization*, pages 109–116, 2000.

[21] S. Schmidt, O. Schögl, R. Kirchberger, H. Doleisch, P. Muigg, H. Hauser, M. Grabner, A. Bornik, and D. Schmalstieg. Novel visualization and interaction techniques for gaining insight into fluid dynamics in internal combustion engines. In *Proceedings of the NAFEMS Worldcongress*, page full Proceedings on CDROM, 2005.

[22] P. Shirley and A. A. Tuchman. Polygonal approximation to direct scalar volume rendering. In *Proceedings San Diego Workshop on Volume Visualization, Computer Graphics*, volume 24, pages 63–70, 1990.

[23] C. T. Silva, J. S. B. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *Proc. IEEE Symposium on Volume Visualization '98 (VolVis '98)*, pages 87–94, 1998.

[24] C. M. Stein, B. G. Becker, and N. L. Max. Sorting and hardware assisted rendering for volume visualization. In *Proc. IEEE Symposium on Volume Visualization '94 (VolVis '94)*, pages 83–89, 1994.

[25] M. Strengert, M. Magallón, D. Weiskopf, S. Guthe, and T. Ertl. Hierarchical visualization and compression of large volume datasets using GPU clusters. In *5th Eurographics/ACM SIGGRAPH Symposium on Parallel Graphics and Visualization (EGPGV 2004)*, pages 41–48, 2004.

[26] M. Trenker, H. Lang, P. Muigg, and H. Doleisch. Validation of vortex flow phenomena in electrical machinery using advanced simulation and visualization techniques. In *Proceedings of the NAFEMS Worldcongress*, page full Proceedings on CDROM, 2007.

[27] D. Uesu, L. Bavoil, S. Fleishman, J. Shepherd, and C. T. Silva. Simplification of unstructured tetrahedral meshes by point sampling. In *Volume Graphics*, pages 157–165, 2005.

[28] M. Weiler and T. Ertl. Hardware-software-balanced resampling for the interactive visualization of unstructured grids. In *Proceedings IEEE Visualization 2001*, 2001.

[29] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. In *Proceedings IEEE Visualization 2003*, pages 333–340, 2003.

[30] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based view-independent cell projection. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):163–175, 2003.

[31] M. Weiler, P. N. Mallón, M. Kraus, and T. Ertl. Texture-encoded tetrahedral strips. In *Proc. IEEE Symposium on Volume Visualization 2004 (VolVis 2004)*, pages 71–78, 2004.

[32] M. Weiler, R. Westermann, C. D. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *Proc. IEEE Symposium on Volume Visualization 2000 (VolVis 2000)*, pages 7–13, 2000.

[33] R. Westermann. The rendering of unstructured grids revisited. In *Proc. of the 3rd Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2001)*, pages 65–74, 2001.

[34] P. L. Williams. Visibility-ordering meshed polyhedra. *ACM Trans. Graph.*, 11(2):103–126, 1992.

[35] Y. Zhou and M. Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE Trans. Vis. Comput. Graph*, 12(5):1229–1236, 2006.