DOI: 10.1111/j.1467-8659.2009.01603.x

EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker (Guest Editors)

Volume 29 (2010), Number 2

Interactive High-Quality Visualization of Higher-Order Finite Elements

Markus Üffinger, Steffen Frey and Thomas Ertl

Visualization Research Center Universität Stuttgart (VISUS), Germany

Abstract

Higher-order finite element methods have emerged as an important discretization scheme for simulation. They are increasingly used in contemporary numerical solvers, generating a new class of data that must be analyzed by scientists and engineers. Currently available visualization tools for this type of data are either batch oriented or limited to certain cell types and polynomial degrees. Other approaches approximate higher-order data by resampling resulting in trade-offs in interactivity and quality. To overcome these limitations, we have developed a distributed visualization system which allows for interactive exploration of non-conforming unstructured grids, resulting from space-time discontinuous Galerkin simulations, in which each cell has its own higher-order polynomial solution. Our system employs GPU-based raycasting for direct volume rendering of complex grids which feature non-convex, curvilinear cells with varying polynomial degree. Frequency-based adaptive sampling accounts for the high variations along rays. For distribution across a GPU cluster, the initial object-space partitioning is determined by cell characteristics like the polynomial degree and is adapted at runtime by a load balancing mechanism. The performance and utility of our system is evaluated for different aeroacoustic simulations involving the propagation of shock fronts.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation—Viewing algorithms, Computer Graphics [I.3.2]: Graphics Systems—Distributed/network graphics

1. Introduction

Discontinuous Galerkin formulations [CKS00] have become increasingly popular since they combine the flexibility in handling complex three-dimensional geometries, hp-adaptivity – refining geometry (h) and increasing the polynomial degree (p) – and the efficiency of parallel implementation in a natural way. Their application ranges from hydrodynamic and aeroacoustic simulations to magnetohydrodynamic calculations of space thrusters or solar winds, in which wave propagation over long distances is desired (Fig. 8). Even highly complex phenomena such as fluid-structure interactions can be handled with these methods.

A significant advantage of these schemes is their capability of performing very high order calculations that need considerably fewer cells to provide a comparable accuracy to classical simulation methods. They are also capable of delivering low dispersion and dissipation errors in complex three-dimensional domains and their solution is, in general, a function with discontinuities at inter-element boundaries while being C^∞-continuous inside the cell. The complexity of our Discontinuous Galerkin data is illustrated in Fig. 1. The adaptivity in the polynomial degree of the solution and the h-adaptive, non-conforming grid is shown.

In contrast to the abilities of simulation systems, an exact rendering of nonlinear functions cannot be directly performed by standard graphics APIs, since they are designed to work with planar primitives that are intrinsically linear. Thus, a widely used approach is to resample the higher-order cells to approximate the polynomial results by means of a considerably larger unstructured tetrahedral grid with linear data [RCMG05, SBM*06]. Other techniques directly visualize the higher order data. But these techniques are either far from being interactive or are limited in the polynomial degree or the type of the underlying grid that can be handled. In particular, there is no visualization technique for direct volume rendering (DVR) of higher-order data with arbitrary polynomial degree. To overcome these limitations, we have developed a GPU raycasting framework which allows for di-

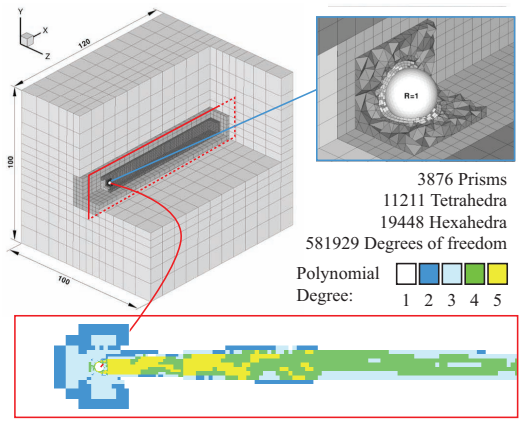

Figure 1: *A non-conforming mesh discretizing the domain of a simulation of fluid flow around a sphere. The adaptive degree of the polynomial solution is depicted on a planar cut marked in the 3D illustration.*

rect visualization of the high-order solution obtained with discontinuous Galerkin methods (see Section 2). In this work our focus is on direct volume rendering of polynomial scalar fields calculated on complex non-conforming grids featuring elements with curved cubic faces.

This paper presents our parallel visualization system with its CUDA raycasting core in detail. A compact monomial representation of the polynomial data is used for efficient storage and evaluation on the GPU. Its benefits compared to an orthogonal polynomial basis representation are discussed. Due to the high order of the polynomials, the function evaluation at each sampling point during the computation of the visualization is very expensive. An adaptive sampling strategy which calculates a conservative approximation of the required sampling rate on a per cell basis reflects the adaptivity in the data to optimize rendering time and to achieve high-quality renderings. Often a single PC with one GPU is still not sufficient to achieve interactive frame rates. We describe an object-space parallelization strategy to distribute the rendering across a cluster. The presented techniques are evaluated by means of two simulation datasets on a stand-alone PC and a GPU cluster with 16 nodes.

2. Space-time Discontinuous Galerkin data

Our hp-adaptive higher-order data is generated by the space-time expansion Discontinuous Galerkin (DG) simulations presented in [GLM08]. For an overview and review of the development of DG methods see [CKS00]. The scalar field solution is represented by piece-wise polynomial solutions of high order with discontinuities at the cell boundaries. The hp-adaptivity of the method implies varying cell sizes and polynomial degree n. The local solution within one cell can

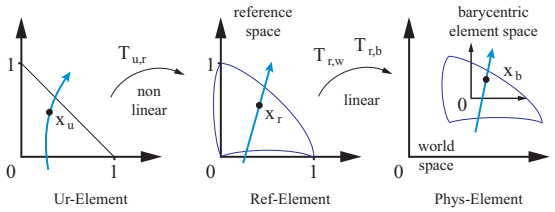

Figure 2: *The element spaces used by the DG simulation. $T_{r,w}$ and $T_{r,b}$ map the reference space element onto its physical space geometry, with respect to the world-space and the element's barycentric coordinate system.*

be written as

$$P(\mathbf{x}) = \sum_i c_i \Psi_i^{(n)}(\mathbf{x}), \tag{1}$$

with $\{\Psi_i^{(n)}\}$ being a basis of the space of polynomials up to degree n and c_i being the corresponding coefficients. To optimize the scheme's efficiency, the simulation uses an orthonormal basis $\tilde{\Psi}_i$ constructed from a monomial basis $\{b_i^{(n)}\} = \{x^u y^v z^w | u,v,w \in \mathbb{N}_0, u+v+w \le n\}$ with $b_i^{(n)}$ given in order of increasing degree (see *coeffID* in Listing 1). Orthogonalization is done with the Gram-Schmidt algorithm

$$\begin{aligned} \Phi_i(\mathbf{x}) &= b_i(\mathbf{x}) - \sum_{j=0}^{i-1} \langle b_i(\mathbf{x}), \tilde{\Psi}_j(\mathbf{x}) \rangle \tilde{\Psi}_j(\mathbf{x}) \\ \tilde{\Psi}_i(\mathbf{x}) &= \frac{\Phi_i(\mathbf{x})}{\sqrt{\langle \Phi_i(\mathbf{x}), \Phi_i(\mathbf{x}) \rangle}}, \end{aligned} \tag{2}$$

where Φ_i denotes an intermediate state and $\tilde{\Psi}_i$ the orthonormalized basis polynomials. Orthonormality of two basis functions is defined as $\langle \varphi_i, \varphi_j \rangle = \delta_{i,j}$, which involves the L_2 inner product $\langle \varphi_i, \varphi_j \rangle := \int_\Omega \varphi_i(\mathbf{x}) \varphi_j(\mathbf{x}) d\mathbf{x}$. Thus, in contrast to the monomial basis, the structure of the orthogonal basis depends on the geometrical area Ω of the cell. To alleviate this overhead, the simulation represents the solution as $P_r(\mathbf{x}_r)$ in the reference space of each cell, where for each type of non-curved elements, e.g., all tetrahedra, the same orthogonal basis $\tilde{\Psi}_i$ can be used. The linear geometry transformation $T_{r,w}(\mathbf{x}_r) = A\mathbf{x}_r + \mathbf{t}_w = \mathbf{x}_w$ maps the reference space element onto its world-space geometry (see Fig. 2).

To visualize the solution, $T_{r,w}$ has to be inverted. For numerical reasons we transform the reference space solution to a barycentric coordinate system local to the cell instead of using the world space system. The corresponding mapping is denoted with $T_{r,b}$. With $P_b(\mathbf{x}_b) := P_r(A^{-1}(\mathbf{x}_b - \mathbf{t}_b))$ the physical space polynomial with unchanged degree can easily be obtained. Our interactive visualization system uses a monomial representation to allow efficient evaluation on the GPU (see Section 6.2 for a discussion) which is obtained by expansion and subsequent gathering of the monomial terms:

$$P_b(\mathbf{x}_b) = \sum_{u+v+w<n} c_{u,v,w} \; x^u y^v z^w. \tag{3}$$

3. Related work

Direct volume rendering of low-order unstructured grids is extensively handled in the literature [MFS06]. We adopt the raycasting approach of Garrity [Gar90], which is able to handle complex unstructured grids, for our system. Visualization algorithms for higher-order basis functions of arbitrary degrees, which are given on arbitrary cell types, have limited coverage in the literature and are still an open research problem. The lack of general higher-order visualization tools is also due to the large number of available finite element (FEM) methods that often use their own proprietary basis functions and element mappings from reference space to physical space.

There are a number of techniques that adaptively subdivide higher-order meshes to generate an optimal linear grid that can be rendered with standard visualization tools. Usually, error metrics control the subdivision depth. In [RCMG05], adaptive mesh refinement techniques are employed to subdivide higher-order elements in reference space. Schroeder and Thompson developed an edge-based and a topology based adaptive tessellation method [SBM*06]. Tesselation is an expensive preprocessing step resulting in an immense increase of data size. Our system belongs to the group of techniques that directly visualize higher-order data. This requires the visualization system to project the polynomial solution, often given in reference space, to the physical world space. The main issue is the costly inverse mapping of positions from world space to reference space that has to be calculated during the visualization process. There is no analytical solution for curved elements, in general. Williams et al. [WMS98] visualize quadratic tetrahedra with flat faces resulting from FEM simulations with DVR accurately. Other cell types can also be handled, but not in high quality. Curved tetrahedra are handled by Wiley's raycasting and isocontouring techniques [WCHJ04, WCG*03]. Parametric curves are fitted to the curved rays in reference space for DVR. Subsequently the polynomial solution is sampled along the curve. Such an approach is limited to quadratic elements. Pixel-exact rendering of isosurfaces for spectral hp-elements is discussed in [NK06]. The 3D reference space solution is projected onto 1D polynomials defined along the viewing rays in physical space. Then, the ray-isosurface intersections are identified by solving a 1D root finding problem for each ray. Those techniques are in general not interactive. A hardware accelerated rendering approach for 2D slicing of 3D space-time cubic tetrahedral grids is discussed in [ZGH04]. Here, the element mappings of the tetrahedra are linear, allowing a fast inversion and evaluation of the solution in reference space. Meshless approximation methods are the third class of higher-order visualization techniques. Zhou and Garland [ZG06] implemented a point-based volume visualization system that resamples non-conforming, tetrahedral meshes with points by applying Lloyd relaxation in a preprocessing step. In [MNKW07], a particle system is proposed to approximate higher-order finite element isosurfaces in reference space. Typically, there is a trade-off between required pre-computation time and desired accuracy of the approximation with this class of methods.

Ma [Ma95] proposed a distributed architecture for tetrahedral grids based on raycasting. For distributed rendering across a cluster, they create object space partitions in a preprocessing step that are balanced concerning the amount of cells they contain. Compositing works on the basis of ray segments that are generated every time the ray enters its sub-volume, which can occur several times due to the concavity of a sub-volume. Vo et al. [VCS*07] used an image-space partitioning technique instead. A distributed volume visualization system based upon kd-trees for object-space partitioning, like we use it in our system, was described by Müller et al. [MSE07] in the the context of uniform grids. They implemented and evaluated static as well as dynamic load-balancing strategies. Aykanat et al. [ACFK07] discuss the object-space decomposition problem for tetrahedral grids in terms of a graph partitioning problem using an estimation scheme for view-dependent node and edge weighting.

4. Raycasting h-adaptive Discontinuous Galerkin grids

The Discontinuous Galerkin scheme discretizes the simulation domain using complex non-conforming unstructured grids consisting of different types of cells. Due to the h-adaptivity of the grid, the sizes of the individual cells can vary over several orders of magnitude. Additionally, the grids contain non-convex regions. Fig. 1 shows a typical example of such a mesh. The dataset contains 1290 curved cubic triangle faces, given in a parametric surface representation in physical space, to better approximate the sphere obstacle within the domain. Non-conforming curved faces are not contained in our data.

There are two major classes of volume rendering techniques for visualizing volume data on unstructured grids. The first class are cell-projection techniques that require the cells to be sorted in visibility order. The cells are then projected to the image plane one after the other. With this approach non-convex cells cannot be handled. Additionally, even for unstructured grids containing convex cells only, determining a correct sorting of the cells is not always possible due to visibility cycles [KE01]. Our system is based on raycasting which is well-suited for these unstructured grids. By exploiting neighborhood information the unstructured grid can be traversed along the viewing rays [Gar90]. At first the entry point of the ray into the grid has to be determined. Often, a search data structure is employed to find the entry cell efficiently. As soon as the entry cell is known, the exit face, where the ray leaves the cell, can be determined. Then, the grid traversal algorithm can directly continue with the next cell along the ray. This also works with curved element faces, given that an appropriate intersection routine is available. Fig. 3 (left) illustrates this by an example. A ray ($r2$)

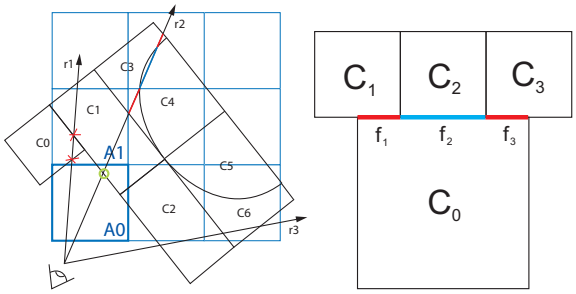

Figure 3: *Unstructured grid featuring curved cells and the atom grid acceleration data structure which is used for finding entry cells and skipping empty regions (left). Non-conforming faces have to be dissected into sub-faces that are shared by at most two cells (right).*

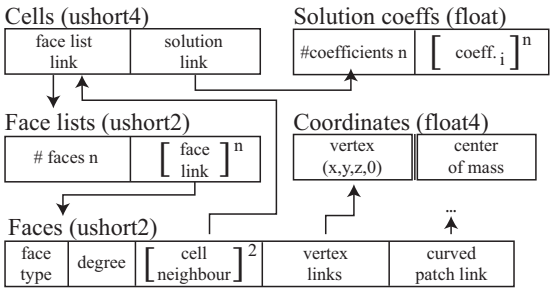

Figure 5: *Each block shows the layout of one data element stored in the 2D texture of the annotated name and format. To be able to refer to arbitrary locations in other 2D textures links are stored in an ushort2 format. Each cell has one link referring to the list of its faces and another link pointing to its field polynomial solution. For each polynomial the number of coefficients n is stored followed by the sequence of the n coefficients.*

traverses a curved cell for a short segment ($C3$, red), leaves the cell, traverses another cell ($C4$, blue), and then re-enters the first cell ($C3$, red). As required by DVR the contributions of the three segments to the final image can easily be blended together in visibility order.

5. Higher-order raycasting system

In this section we describe the rendering system including the raycasting kernel implemented in CUDA. Parallelization is discussed in Section 7. An overview diagram of the system is shown in Fig. 4.

After loading the higher-order dataset three preprocessing steps are performed. To speed up the procedure of finding the cells where the rays enter the grid, an acceleration data structure (atom grid) is constructed. Secondly, bounding volumes are generated for curved faces. Thirdly, a gradient analysis of the polynomial solution is performed, which is needed for the adaption of the sampling step size (see Section 6.4). The steps are executed only once for each dataset. The topology and geometry of the unstructured grid and the polynomial data are stored in multiple 2D textures to benefit from texture cache. Fig. 5 illustrates the data layout and inter-texture dependencies. To store the coefficients of the polynomial solution a 2D texture is used due to the size limitations of 1D textures. The atom grid is stored in a 3D texture. For the transfer function a 1D texture is sufficient. The textures are uploaded to the GPU before the main render loop starts.

A frequency analysis, which is needed by the step size adaption algorithm, is performed each time the transfer function changes. The sampling step sizes are calculated for each cell, and uploaded to the GPU, altogether taking less than one second. Then the raycasting kernel is executed. For each pixel of the resulting image, a ray is cast into the scene starting at the camera location.

The raycasting kernel consists of three major phases. In the initialization phase, the ray is constructed and the intersection of the ray with the dataset's bounding box is calculated. The ray thread stops immediately if there is no intersection, otherwise the second phase (Atom Grid Phase) and third phase (Grid Traversal Phase) are executed in an iterative manner, until the ray leaves the bounding box. In order to find the entry cell efficiently, to be able to handle non-convex grids with holes and to partition the domain for parallelization, we employ a uniform grid structure. This atom grid resides above the actual simulation grid and is traversed in the Atom Grid Phase. Each atom grid cell contains links to the dataset's cells it covers. The edge length of the smallest cell defines the desirable resolution of the atom grid. If the grid does not fit into graphics memory with this resolution, it is coarsened globally. For determining the entry cell for a given ray, regardless of whether this is the initial entry cell or whether the grid has been left before, the ray is checked against all cell faces of the current atom grid cell. If there is an intersection, a new cell entry position has been found, otherwise it proceeds to the next atom grid cell according to the 3D voxel traversal algorithm [AW87]. The ray entry search is illustrated in Fig. 3 (left). Ray $r2$ finds its entry cell $C1$ in atom $A0$. The intersection of ray $r1$ with $C0$ and $C1$ however are invalid in $A0$. The entry cell $C0$ of $r1$ is found after traversing to $A1$.

If an entry cell and position is found, the grid traversal phase starts. At first the exit face and position where the ray leaves the cell is determined. Based on the entry and exit position the segment of the ray within the cell is known. Now, the higher-order polynomial field solution of the cell is sampled along this ray segment. Section 6 describes the higher-order polynomials, the storage, and the evaluation in the kernel. After a cell has been processed, the algorithm goes on iteratively with the next cell found along the ray. If the ray leaves a cell through a face without an adjacent neighbor cell, the algorithm proceeds with the Atom Grid Phase.

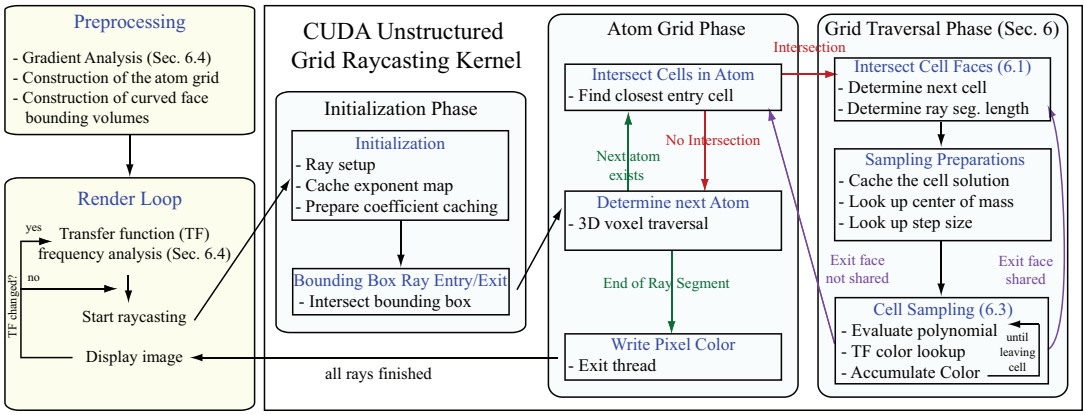

Figure 4: *Overview diagram of the system. Alternative execution paths are shown in one color.*

6. Grid traversal phase

The raycaster handles hybrid grids that consist of different types of cells, including tetrahedra, hexahedra, prisms and pyramids. Non-conforming cells, where a cell can have more than one neighbor directly accessible through one face of the cell, are also supported. Our raycaster expects faces shared by two neighbors at most. Thus, non-conforming faces have to be dissected as in Fig. 3 (right). In our case the dissection is already handled by the simulation.

6.1. Ray curved face intersection

The raycasting system supports higher-order elements with curved faces. Our data contains cubic quad and triangle faces that are given in a parametric surface representation. Ray patch intersection routines are integrated into the kernel for those patches. The system can be extended to other types of patches in a straightforward manner. The intersection of the ray with a cubic patch cannot be calculated analytically. A common approach is to represent the ray by two intersecting planes and employ an iterative Newton-Raphson solver to find the roots of the resulting two non-linear equations [GA05]. A good initial guess as a starting point for the iteration is required for fast convergence. In raytracing bounding volume hierarchies are used to isolate the roots prior to applying Newton-Raphson. We use object-aligned parallelepipeds for the quad faces and tripipeds for the cubic triangles [BS93]. Fortunately, our curved faces are only slightly curved; a single bounding volume is sufficient for the algorithm to converge to the intersection point in less than 4 Newton iterations. If the angle between the ray and the patch becomes too small, the ray may intersect the patch at multiple points. In such cases the Newton iteration is started at two opposite positions on the patch. Then the correct intersection point can be isolated [BS93]. In the CUDA kernel the patches are represented in a monomial basis. The quad faces, which are cubic tensor-product patches, are represented by 16 3D vector coefficients. For the triangle patches 10 vector coefficients are required. The coefficients of the patches are stored in a 2D texture. For the calculation of the bounding volumes a Bézier representation is used.

6.2. Polynomial representation

The raycasting system samples the barycentric monomial representation $P_b(\mathbf{x}_b)$ in physical space (see Section 2). By not switching to reference space the transformation $T_{r,b}^{-1}$ does not need to be performed at run time. A compact representation of the polynomial solution is obligatory for efficient evaluation. We additionally need the gradient of the field at the sample points for shading calculations. The monomial representation meets our requirements best. Gradients come at low extra cost and the basis is compact in the sense that only the coefficients $c_{u,v,w}$ need to be stored explicitly (see Eq. 3). The exponents u,v,w of the respective monomial basis functions can be determined according to the scheme given in Listing 1. We illustrate the difference in size of the representations by the example of the simulation's Gram-Schmidt orthogonalized basis, built from monomials. Here, the structure of the basis depends on the coefficients $\langle b_i(\mathbf{x}), \tilde{\Psi}_j(\mathbf{x})\rangle$. A formula for the number of these coefficients can be directly derived from the algorithm of Eq. 2 with an arithmetic series. Table 1 shows that with increasing degree n the number of total coefficients $o(n)$ of the or-

Table 1: *Comparison of the number of coefficients m required to represent the polynomial solution of degree n of a general curved element in the monomial basis and the number of coefficients o required in the orthogonal basis.*

degree n	3	4	5	6	7
$m(n) = \frac{(n+1)(n+2)(n+3)}{6}$	20	35	56	84	120
$o(n) = \frac{m(n)(m(n)+1)}{2}$	210	630	1596	3570	7260
overhead factor	10.5	18	28.5	42.5	60.5

thogonal polynomial quickly becomes very large. Note, $o(n)$ represents the worst case scenario of a general (curved) cell where all coefficients are non-zero.

The missing orthogonality property of the monomial basis functions makes it unsuitable for simulation since it does not yield sparsely populated matrices which allow efficient iterative solvers to be used on the resulting system of equations. Moreover, it introduces numerical instabilities due to badly conditioned matrices. Our visualization system however only needs to evaluate the resulting final solution. The calculations are done with single precision floating-point accuracy on the GPU. Being concerned with accuracy, we investigated the numerical stability. For our datasets a double precision evaluation, which we took as reference, revealed a very low maximum relative point-wise error of $1.3 \cdot 10^{-6}$ for the evaluation of the monomial representation in barycentric coordinates with single precision.

6.3. Evaluating the polynomials in the kernel

To evaluate the monomial representation at a point, its world coordinates are transformed to the cell's local barycentric coordinate system. The local coordinate is then used to iteratively evaluate the polynomial and the gradient, as outlined in Listing 2. The scheme in Listing 1 defines an increasing degree order for the monomial basis functions, enabling the support for polynomials of varying degree. As the exponents are the same for all cells, we precompute them up to degree seven, and load them into the 1D array *map* stored in fast shared memory before raycasting starts. Caching is the better choice than a calculation at runtime since the coefficients only require 360 Bytes (120 unsigned bytes for each x,y, and z). Due to the size limitations of 1D textures, the degrees and coefficients of all polynomials are stored in one large 2D texture, each cell occupying a small consecutive block of memory that contains its coefficients in this order. We preload the coefficients of the polynomial into fast shared memory before the sampling. As the rays of one block are largely incoherent, each ray thread needs to load the polynomial coefficients of its cell. The number of coefficients that can be preloaded is limited by the number of ray threads in the block and the available shared memory.

```
int coeffID=0;
float3 map[m(n)]; // exponent array of size m(n) (Table 1)
for (int i=1; i<=n+1; i++)
 for (int j=1; j<=i; j++)
  for (int k=1; k<=i−j+1; k++) {
   map[coeffID].x = i−j−k+1; // u, exponent of x
   map[coeffID].y = j−1; // v, exponent of y
   map[coeffID].z = k−1; // w, exponent of z
   coeffID++;
  }
```

Listing 1: *Pseudocode to compute the exponents of the 3D monomial basis for a given maximum polynomial degree n.*

6.4. Adaptive sampling for direct volume rendering

With a conservative choice of a user-defined 'safe' minimum sampling step size for the whole dataset large parts of our hp-adaptive data gets oversampled, leading to a very large increase in runtime, whereas on the other hand it can not be guaranteed that with the chosen rate all features are sampled adequately. Calculating the required sampling rate is not trivial, as the function to sample is not known a priori. DVR usually has to deal with a composited color function $k_c(\mathbf{x}) = g \circ f(\mathbf{x})$, which determines the color k_c of a field sample by applying a user defined transfer function g to the scalar field sample $f(\mathbf{x})$. However, it is not practical to determine the exact sampling frequency every time a sample is taken. Thus, in our system, a semi-conservative sampling strategy on a per-cell basis is applied, sampling each cell with its specific adapted sampling rate. The required sampling rate depends on the color function k_c. According to Bergner *et. al* [BMWM06] it is possible to analyze the composited signal $k_c = g \circ f$ by looking at both components separately. A good approximation of the local band limiting frequency $\nu_{k_c} = \nu_g \max_{\mathbf{x}} |f'(\mathbf{x})|$ can be calculated by taking the maximum frequency ν_g of the transfer function times the local gradient magnitude of the underlying scalar field. In a preprocessing step we determine for each cell the range of field values $[f_{\min}, f_{\max}]$ and the maximum gradient magnitude $|\nabla f|_{\max}$ within the cell by sampling the cell's polynomial solution on a very fine grid. Each cell is sampled with ten times the polynomial degree samples in each direction.

The frequency analysis of our transfer function, has to be

```
void evalPolynomial(float3 pos, float& res, float3& grad) {
 res = 0; // scalar field sample
 grad = {0,0,0}; // gradient field sample

 for (int coeffID=0; coeffID<m(polynomialDegree); coeffID++) {
   float3 prod = pow(pos, map[coeffID]); // evaluate monomial
   prod = prod.x∗prod.y∗prod.z; // basis function

   float coeff; // fetch basis function coefficient
   if (coeffID < MAX_COEFFS_IN_SHARED_MEM)
    coeff = sharedMemoryCoeffs[coeffID];
   else
    coeff = texture2D(texCoeffs, solutionOffset+coeffID);

   res += coeff∗prod; // add monomial contribution
   grad.x += map[coeffID].x ∗ (coeff∗prod/pos.x);
   grad.y += map[coeffID].y ∗ (coeff∗prod/pos.y);
   grad.z += map[coeffID].z ∗ (coeff∗prod/pos.z);
  }
}
```

Listing 2: *Evaluating the field polynomial on the GPU.*

Figure 8: *Illuminated direct volume rendering of the sphere dataset showing the typical von Karman vortex street roll-up.*

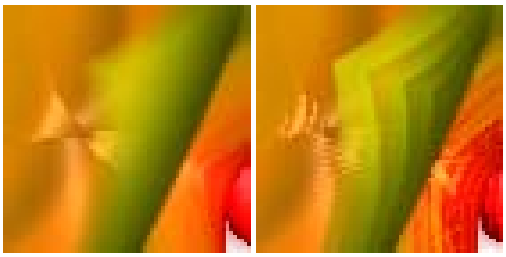

Figure 6: *Image crop of the shock channel results (see Fig. 11) showing the discontinuities in the data and the effect of trading rendering quality against responsiveness on a stand-alone PC. High quality rendering (left, step size factor 1, 6.3 sec), interactive rendering (right, factor 20, 0.4 sec).*

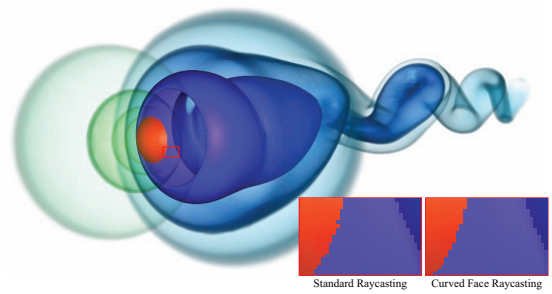

Figure 7: *Direct volume rendering of the flow around the sphere (large image). Only directly on the curved sphere a difference between raycasting with and without curved face handling is visible (small images).*

executed each time the user modifies the color mapping of the scalar field. In our case the user is able to draw RGBA transfer functions that are stored in an array of size 256. Fast Fourier Transform is employed to calculate the frequency power spectra of the associated color functions $(R,G,B)\times A$. Cells often only cover a small part $[f_{\min}, f_{\max}]$ of the transfer function's range. We are exclusively interested in the frequencies occurring in the cell's narrow window. To account for this, the part of interest is cut out of the transfer function and resampled again onto 256 samples prior to applying the FFT. In general, the windowed part of the transfer function is non-periodic. To prevent artificial high frequency components introduced at the borders, the transfer function is multiplied by a Hanning-Window $w(x) = 0.5 - 0.5cos(2\pi x/d)$, where d is the width of the transfer function's range. In frequency space this results in a convolution of g_f with the window's spectrum w_f, which approximately equals a smoothing with a symmetric Gaussian-like kernel of width three. Because of that the direct current (DC) component of the signal should be removed before applying the windowing function. Another issue comes with the hand drawn non-smooth nature of the transfer function. To get reasonable results from the frequency analysis, the functions are smoothed with a Gaussian kernel before transforming to frequency space. The color channel with the highest frequency component, larger than a small threshold, determines ν_g. With all components at hand ν_{k_c} can be determined for each cell and the step size $h = 0.5/\nu_{k_c}$ can be calculated. Multiplying the adaptive step size h with a factor smaller than 1 improves the overall rendering quality. To improve responsiveness during camera interaction a factor larger than 1 can be chosen (Fig. 6).

7. Parallelization

We use an object-space partitioning approach to distribute raycasting across multiple GPUs in a cluster environment. The grid is subdivided into as many partitions as there are GPUs. Note that in our case, unlike for medical volume data, the performance for datasets that can be visualized in high quality close to interactivity is limited by pure rendering speed only, while GPU memory capacity does not play a big role. Thus, we limit ourselves to task distribution and do not consider data distribution in this work, which means that the whole dataset with all the necessary data structures needs to fit onto each single GPU. However, this also means that data transfers at runtime are avoided which potentially have a large impact on load balancing. Subdividing the dataset on cell level would be problematic as this might lead to non-convex partitions which makes correct compositing impossible. Instead, we use the overlaying uniform atom grid structure that we initially introduced for determining entry cells. Atoms cover a cuboid volume and contain cells that are included or intersect this volume.

In order to distribute the rendering task across multiple nodes, our system partitions the atom grid to convex, non-overlapping groups of atoms called bricks. A brick is rendered on the GPU in one pass and the corresponding rays are clipped to that brick. This guarantees that no regions of cells that are contained in multiple bricks are sampled several times. The renderings of all nodes are transferred to the display node, that usually is also used for user interaction, to composite the final image using standard blending for each frame. Backend nodes determine the blending order of their corresponding brick by considering the view point and us-

ing the kd-tree which is employed for load-balancing as explained below. Finally, the order index is attached to the image before sending it to the frontend node.

For the initial partitioning of the atom grid into bricks and additionally for supporting dynamic load balancing at runtime, each atom saves a value that roughly estimates its rendering complexity. It is computed by considering all contained cells q_i and summing up their weighted geometry $l(type(q_i))$ and the number of monomials $m(deg(q_i))$ (see Tab. 1) of their polynomial solution, whereas the influence of a cell is relative to its volume inside the atom $vol(q_i)$:

$$\sum_{i=1}^{\#cells} vol(q_i) \cdot (l(type(q_i)) + m(deg(q_i))). \qquad (4)$$

The geometry weight function l returns the highest values for cells featuring curved faces. Besides this static complexity measure, the load balancing algorithm also considers the rendering time of each brick which is distributed over all atoms of the brick relative to their complexity values. Optionally, the user can manually tweak the performance by adjusting the weighting of the complexity components and the influence of the complexity measure in the rendering time distribution. The estimated render times per atom are used to determine which atoms to add or to remove from a brick and thus influence the workload of a GPU.

The partitioning of the atom grid is done using a kd-tree which is rebalanced every frame. It is traversed top-down and sub-trees are balanced by moving the split plane towards the region with the larger overall estimated render time until the optimal solution is found. This means that the weights on both sides should be as close as possible. Moving the split plane is slowed down by a rubber band effect that penalizes the relocating of the split plane. It weights the overall estimated render time of the smaller partition with a factor that increases with the number of atoms that the split plane moved and can be adjusted by the user. Slowing down avoids cases in which huge back and forth leaps occur per frame due to an inappropriate complexity estimation

8. Results

Our higher-order rendering framework allows scientists to interactively visualize data of their higher-order simulations in contrast to the traditional time-consuming approach of resampling and using standard postprocessing software. Using our adaptive sampling approach, rendering times for high-quality visualizations are in the range of one second up to 20 seconds on a stand-alone PC. Fig. 6 demonstrates how render quality can be reduced during camera movements to achieve interactive framerates in such an environment. As soon as the camera movement stops, the high quality image of Fig. 11 is calculated and presented to the scientist. In a cluster environment interactive rates can be obtained even with the high quality settings.

Table 2: *Relation of render time to solution degree n showing a linear scaling in the number of basis functions $m(n)$.*

n / $m(n)$	2 / 10	3 / 20	4 / 35	5 / 56	6 / 84
t in ms	205	307	476	730	1067

The performance measurements have been conducted for two application datasets. The sphere test-case is a hydrodynamical simulation solving the compressible Navier-Stokes equations: with a Reynolds number of $Re = 300$ a uniform flow of Mach number $Ma = 0.3$ is initially set up. The simulation should then show a typical von Karman vortex street roll-up as illustrated in Fig. 8. The shock channel dataset shown in Fig. 11, is a numerical simulation where a $Ma = 3$ shock hits an obstacle positioned in the middle of a channel. As the shock moves over the obstacle, a lifted ballistic wave should form together with reflections of the shock wave on the channel walls. The handling of shocks as well as a sufficient resolution of the effects can be a challenging task for low order numerical schemes. In the shock channel simulation the high polynomial degree of seven compensates nicely for the very low resolution ($20 \times 2 \times 3$) of the underlying regular mesh. Here, the box obstacle is the size of one cell. Some stronger discontinuities at the boundaries of the cells, which are inherent to the DG solution, can be seen in the closeup views in Fig. 6. The red isoslab of the upper left cell is strongly curved in the lower right edge, not matching the solution of the neighboring cells. In the direct vicinity of the obstacle in the sphere dataset the mesh consists of 1290 curved cubic triangles. Timings are given in Table 3 for raycasting with handling of curved faces enabled and disabled for the sphere dataset. Fig. 7 shows the improved image quality that is achieved with curved face handling enabled. The red high density regions touching the front of the sphere better reflects the curved surface of the obstacle. We generated an artificial dataset to determine the scaling of the performance with the polynomial degree of the data (Table 2). To compare our higher-order raycasting image results, we resampled the higher-order data onto a regular grid and visualized it with a standard GPU-based raycaster. Fig. 10 illustrates that even a very high resampling resolution, which leads to a dataset size that can not be handled by a single PC, is by far not high enough to capture all features in the data.

For the evaluation of the scaling of our distributed system, up to sixteen cluster nodes were used that are equipped with a NVIDIA GeForce 285 GTX each and connected via Gigabit ethernet. A series of 285 frames was rendered while moving the camera around the dataset and zooming in and out at various speed. The averaged timing results over all frames are depicted in Fig. 9. It can be seen that the system overall exhibits good scaling behavior. However, it is hindered by several effects to a different degree. As can be seen in the left graph, the total rendering time is close to the max-

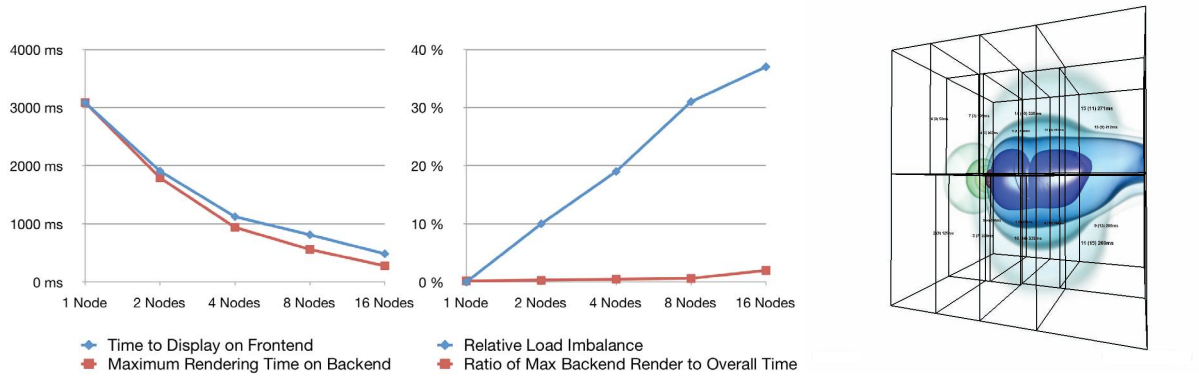

Figure 9: *Left: Scaling of performance with the number of rendering nodes. Middle: Two effects hindering balancing. The blue line depicts the sum of the ratio of all backend rendering times to the maximum backend rendering time for each timestep:* $\sum_{i=1}^{timesteps} \frac{max(times)-times[i]}{(timesteps-1)\cdot max(times)}$. *This shows total backend rendering imbalance. The red line illustrates the relative difference between the maximum backend render time and the time it takes for the frontend node until display. It can be seen that what hinders better scaling is primarily load imbalance rather than network traffic. Right: Kd-tree of a frame on the camera path.*

imum rendering time on the backend nodes which induces that the impact that network traffic has on scaling is quite small, even though it naturally increases with the amount of rendering nodes involved. This shows more clearly in the red line of the second graph which illustrates that there is only a small difference between the maximum backend render time and the overall time to display. In contrast, the blue line of the second graph depicts significant load imbalance between the backend nodes which results in a lower occupancy of the rendering nodes. The reason for that is that we have a very inhomomogeneous dataset with widely varying complexities but a uniform grid on which load balancing is based. Only few atoms cover a significant share of the overall complexity of the dataset. Thus a big, indivisible part of the dataset is moved in one balancing step, as atoms cannot be split. Furthermore, not only single atoms but a whole 2D array of atoms is added to a brick when moving the split plane of the kd-tree for one unit. Due to this, the dataset cannot be distributed evenly according to its complexity which finally leads to significant variations in the rendering times of the backend nodes. It can also be observed that this limited partitioning granularity leads to bigger performance impacts with a growing number of render nodes due to the increasing distribution imbalance. We performed our tests using an atom grid with the atom grid resolution $288 \times 312 \times 312$.

9. Conclusion and Future Work

The implementation of the proposed visualization system was driven by the fact that traditional visualization software is not able to provide an adequate mechanism to visualize higher-order data resulting from discontinuous Galerkin simulations. We showed the advantages of our system. The time required to generate a visualization of this data in comparison to common approaches is drastically reduced as the simulation data is visualized directly. High-quality rendering results are achieved with an adaptive sampling technique fitted to the hp-adaptive data which calculates appropriate sampling step sizes on a per cell basis. Moreover, our system exploits the computational power of GPU clusters. For future work we consider the replacement of the uniform atom grid with a more adaptive structure like a kd-tree that allows to better adapt to the inhomogeneous properties of the dataset. This would allow for a more efficient determination of entry cells and significantly improve scaling. Additionally, we want to extend the system to data distribution for handling even larger datasets.

Acknowledgements

We thank our colleagues from the Institut für Aero- und Gasdynamik for their continuous support and for providing datasets. This work is supported by Deutsche Forschungsgemeinschaft (DFG) within the Cluster of Excellence in Simulation Technology.

References

[ACFK07] AYKANAT C., CAMBAZOGLU B. B., FINDIK F., KURC T.: Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids. *J. Parallel Distrib. Comput. 67*, 1 (2007), 77–99.

[AW87] AMANATIDES J., WOO A.: A fast voxel traversal algorithm for ray tracing. In *Eurographics* (1987), pp. 3–10.

[BMWM06] BERGNER S., MÖLLER T., WEISKOPF D., MURAKI D. J.: A spectral analysis of function composition and its implications for sampling in direct volume visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1353–1360.

[BS93] BARTH W., STÜRZLINGER W.: Efficient ray tracing for Bézier and B-spline surfaces. *Computers & Graphics 17*, 4 (1993), 423–430.

[CKS00] COCKBURN B., KARNIADAKIS G. E., SHU C.-W.: *Discontinuous Galerkin Methods*. Lecture Notes in Computational Science and Engineering. Springer, 2000.

[GA05] GEIMER M., ABERT O.: Interactive Ray Tracing of Trimmed Bicubic Bézier Surfaces without Triangulation. In *Proc. of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)* (Feb. 2005), pp. 71–78.

[Gar90] GARRITY M. P.: Raytracing irregular volume data. In *VVS '90: Proceedings of the 1990 Workshop on Volume Visualization* (1990), ACM, pp. 35–40.

[GLM08] GASSNER G., LÖRCHER F., MUNZ C.-D.: A Discontinuous Galerkin Scheme based on a Space-Time Expansion II. Viscous Flow Equations in Multi Dimensions. *J. Sci. Comput. 34*, 3 (2008), 260–286.

[KE01] KRAUS M., ERTL T.: Cell-Projection of Cyclic Meshes. In *Proceedings of IEEE Visualization '01* (2001), pp. 215–222.

[Ma95] MA K.-L.: Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures. In *PRS '95: Proceedings of the IEEE Symposium on Parallel Rendering* (1995), ACM, pp. 23–30.

[MFS06] MARMITT G., FRIEDRICH H., SLUSALLEK P.: Interactive Volume Rendering with Ray Tracing. In *Eurographics State of the Art Reports* (2006), pp. 115–136.

[MNKW07] MEYER M., NELSON B., KIRBY R., WHITAKER R.: Particle systems for efficient and accurate high-order finite element visualization. *IEEE Transactions on Visualization and Computer Graphics 13*, 5 (2007), 1015–1026.

[MSE07] MÜLLER C., STRENGERT M., ERTL T.: Adaptive load balancing for raycasting of non-uniformly bricked volumes. *Parallel Comput. 33*, 6 (2007), 406–419.

[NK06] NELSON B., KIRBY R. M.: Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE Transactions on Visualization and Computer Graphics 12*, 1 (2006), 114–125.

[RCMG05] REMACLE J.-F., CHEVAUGEON N., MARCHANDISE E., GEUZAINE C.: Efficient visualization of high-order finite elements. *Journal for Numerical Methods in Engineering 69*, 4 (2005), 750–771.

[SBM*06] SCHROEDER W. J., BERTEL F., MALATERRE M., THOMPSON D., PEBAY P. P., O'BARA R., TENDULKAR S.: Methods and framework for visualizing higher-order finite elements. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (2006), 446–460.

[VCS*07] VO H., CALLAHAN S., SMITH N., SILVA C., MARTIN W., OWEN D., WEINSTEIN D.: iRun: Interactive Rendering of Large Unstructured Grids. In *Proceedings of the 7th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2007)* (2007), pp. 93–100.

[WCG*03] WILEY D. F., CHILDS H. R., GREGORSKI B. F., HAMANN B., JOY K. I.: Contouring curved quadratic elements. In *VISSYM '03: Proceedings of the Symposium on Data Visualisation* (2003), pp. 167–176.

[WCHJ04] WILEY D. F., CHILDS H. R., HAMANN B., JOY K. I.: Ray casting curved-quadratic elements. In *VISSYM '04: Proceedings of the Symposium on Data Visualisation* (2004), pp. 201–210.

[WMS98] WILLIAMS P. L., MAX N. L., STEIN C. M.: A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (1998), 37–54.

[ZG06] ZHOU Y., GARLAND M.: Interactive point-based rendering of higher-order tetrahedral data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1229–1236.

[ZGH04] ZHOU Y., GARLAND M., HABER R.: Pixel-exact rendering of spacetime finite element solutions. In *Proceedings of IEEE Visualization* (2004), pp. 425–432.

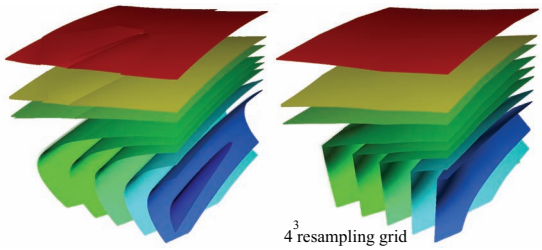

Figure 10: *Comparison of higher-order raycasting (left) and DVR of brute-force resampled data on a regular grid (right). A small region of the sphere dataset with an edge length of* 0.3 *in comparison to a size of* 120 *for the whole dataset is shown. With the chosen resolution the resampled visualization data would require more than 14 GBytes of storage (scalar and gradient field) for the whole domain.*

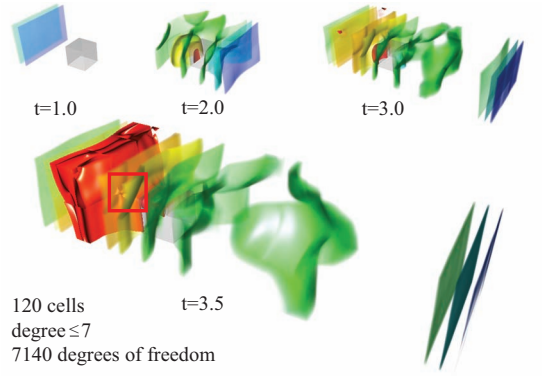

Figure 11: *Shock channel sequence of flow around a box. A zoom-in image of the marked red region is given in Fig. 6.*

Table 3: *Performance results for adaptive and fixed step sizes in seconds measured on a PC with an AMD Opteron with 2.3 GHz, 32 GB of RAM and a NVIDIA GeForce 285 GTX (1024MB) (1280×910 viewport). The global minimum step size of the adaptive case was used for the fixed case.*

	adaptive	fixed	min step size
Channel ($t = 1.0$)	1.01	9.2	0.003
Channel ($t = 2.5$)	2.2	19.8	0.0013
Channel ($t = 3.5$)	6.3	23.2	0.001
Sphere	1.5	>60	0.005
Sphere (curved)	2.0	>60	0.005
Sphere Zoom In	7.9	>60	0.002
Sphere Z. (curved)	11.5	>60	0.002