Direct interactive visualization of locally refined spline volumes for scalar and vector fields

Franz G. Fuchs\textsuperscript{a,}\textsuperscript{*}, Oliver J. D. Barrowclough\textsuperscript{a}, Jon M. Hjelmervik\textsuperscript{a}, Heidi E. I. Dahl\textsuperscript{a}

\textsuperscript{a}SINTEF Digital, Forskningsveien 1, 0314 Oslo, Norway

\textbf{A B S T R A C T}

We present a novel approach enabling interactive visualization of volumetric Locally Refined B-splines (LR-splines). To this end we propose a highly efficient algorithm for direct visualization of scalar and vector fields given by an LR-spline. For the case of scalar fields a volume rendering approach is designed, along with methods for the necessary adaptive sampling distance, on-the-fly trimming with a surface geometry given by a STereoLithography (STL)-file, and local volume illumination based on on-the-fly evaluation of the derivative of the underlying LR-spline function. For vector fields we design a two-stage algorithm consisting of the computation of stream lines with adaptive step size and their rendering with tubes. In both cases, the common basic ingredient to achieve interactive frame rates is an acceleration structure based on a k-d forest together with suitable data structures. The algorithms are designed to fully utilize modern graphics processing unit (GPU) capabilities. Important applications where LR-spline volumes emerge are given for instance by approximation of large-scale simulation and sensor data, and Isogeometric Analysis (IGA). For the first case – approximation of large three dimensional point clouds with an associated field – we provide an extension of the multilevel B-spline approximation (MBA) algorithm to the case of volumetric LR-splines. We showcase interactive rendering achieved by our approach on different representative use cases, stemming from simulations of wind flow around a telescope, Magnetic Resonance (MR) imaging of a human brain, and simulations of a fluidized bed used for mixing and coating particles in industrial processes.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

In recent years, there has been increasing industrial and scientific interest in splines, i.e., piecewise polynomial functions, in higher dimensions, driven by research efforts in managing large heterogeneous data sets and advances in computational science. This has resulted in the development of several independent approaches to local refinement of splines in dimensions higher than one, including hierarchical splines, T-splines, and LR-splines. The main advantage of local refinement is that it focuses approximation power locally where it is needed, allowing a considerable reduction in the amount of data needed to guarantee a given tolerance.

Large heterogeneous data sets can originate from many different sources. One example is geospatial data, where existing and emerging data acquisition techniques provide a fast, efficient and affordable means for data collection. The resulting raw data is generally very large and needs to be represented in a more suitable way for, e.g., comparison and quality assurance. Another example comes from simulations in science, engineering and industry. The design, validation and optimization of products and structures involves, e.g., physical, chemical, or bi-

\textsuperscript{*}Corresponding author: franzgeorgfuchs@gmail.com
ological processes that are modeled by partial differential equations (PDEs). High-fidelity simulations are often necessary in order to perform a reliable analysis. At the time of writing, the computational requirements and scalability of post-processing and visualization tools are a bottleneck in the simulation workflow. The typical size of the simulation data is nowadays on the order of giga- or terabytes per time step, which is both impractical and inefficient to stream or download to a local client. In both these cases locally refined splines have proven to be a good means to compactly represent these data, see, e.g., [1]. A quantitative and qualitative analysis of LR-spline approximations is the focus of an upcoming paper and will not be elaborated in this article.

In computational science a recent development called isogeometric analysis (IGA), proposed by [2], has quickly become very popular in science and engineering. IGA provides the integration of design and analysis by using a common representation – splines – for computer aided design (CAD) and finite element methods (FEM). This eliminates the conversion step between CAD and FEM, which is estimated to take up to 80% of the overall analysis time for complex designs [2]. The use of local refinements of splines is an indispensable requirement for efficient computations. Despite the fact that locally refined splines have become very popular in a variety of situations, methods for visualization of these types of splines are lagging behind or are nonexistent. Direct volume and vector field visualization based on LR-splines is completely novel. In this paper we present a flexible framework for volumetric visualization based on volume and streamline rendering enabling interactive, direct visualization of volumetric LR-splines, that can be trivially extended to T-splines and hierarchical splines. Our approach consists of several stages, leveraging the strengths of existing algorithms where possible. Several features of our approach are novel:

1. For fast evaluation of LR-splines we extend the recently published method presented in [3] for interactive pixel-accurate rendering of LR-spline surfaces to the volumetric case.
2. We compare and discuss different acceleration structures for LR-spline element look-up: textures, octrees, k-d trees, and k-d forests.
3. We present an algorithm for adaptive sampling distance based on the underlying irregular data structure and the degree of the spline, dramatically improving the efficiency of volume and vector field rendering.
4. We reduce the memory footprint of k-d trees by using high bits to store the split direction.
5. We present on-the-fly trimming with STL geometries, and local illumination based on on-the-fly evaluation of the derivative of the underlying LR-spline function for volume rendering.
6. We extend the LR-multilevel B-spline approximation (LR-MBA) method [1] to the case of volumetric LR-splines.
7. We apply the LR-MBA approach to a variety of large data sets in order to generate three-dimensional approximations that are suitable for interactive visualization.

The rest of the paper is organized as follows. Related work is discussed in Section 2, followed by a description of the LR-MBA method in three dimensions, presented in Section 3. Then, an algorithm for efficient evaluation of LR-splines is described in Section 4. Finally, we present applications and provide details of the performance of the implementation of the overall algorithm for interactive volume rendering in Section 5 and vector field visualization in Section 6 and the conclusion in Section 7.

2. Related Work

Scientific volume visualization techniques convey information about a vector or scalar field defined on a given geometry. The techniques can be divided into the following approaches: iso-surface extraction and volume rendering for scalar fields; direct approaches, e.g. glyphs, line integral convolution (LIC), and stream-/streak-/pathline rendering for vector fields. In this article we focus on volume and stream-/streak-/pathline rendering.

The main challenge for achieving interactive volume visualization in the setting of LR-splines is that sampling is computationally expensive due to the need for spline evaluation. However, they allow for a compact representation, particularly when local refinement is applied.

2.1. Scalar Field Visualization

The visualization of a scalar field is commonly done by modeling the scalar field as a participating medium, where a modifiable transfer function specifies how field values are mapped to emitted color and transparency. In the simplest case, where the field consists of discrete samples over a regular grid, an abundance of results is available, see e.g., [4] for an early example or [5] for an overview. High-performance out-of-core methods are one way to tackle large-scale data. The authors in [6] present a high-performance I/O file system to stream time-varying data of the size of 1024³ to achieve target frame rates of 5 fps. Octrees allow a voxel grid to have different resolutions across the domain, which reduces the amount of data needed for a given scene. This is used in e.g., GigaVoxels [7], which offers real-time rendering of several billion voxels. However, there are several limitations to this approach. If the total data size is high the approach is not applicable across networks with limited bandwidth. More importantly, GigaVoxels is only effective when location of significant regions of empty space is known a priori. In our case, where an interactive transfer functions can create empty space, this assumption is prohibited.

This article focuses on visualizing volumetric data on an irregular grid (non-uniform rectilinear, curvilinear, unstructured, or adaptive mesh refinement (AMR) data), which requires additional processing and storage space compared to regular data. In particular, direct rendering needs methods for view-order cell-traversal, and interpolation inside cells. [8] focuses on direct volume rendering of AMR data, but the types of mesh are heavily restricted. High quality rendering of more general AMR data is presented in [9]; the authors report 4 fps on screen resolutions...
up to $2048^2$ on a data set with 2377878 cells with 4 levels of refinement. For a recent survey on state-of-the-art GPU-based large-scale volume visualization we refer to [10].

2.2. Vector Field Visualization

In the case of vector fields, occlusion and complexity make direct visualization of the whole data set very hard to interpret visually. Interactive texture based flow visualization were one of the first use cases for programmable graphics hardware [11]. Such methods take advantage of and rely on fast texture lookups and regular memory operations to obtain good performance, which limits the use to uniform grids.

For glyphs and stream-/streak-/pathline rendering, seeds are typically chosen directly by the user, or are based on simplification or feature extraction. The main requirement for interactive implementation of these techniques is fast evaluation of the vector field. In this paper we will focus on the evaluation algorithm of volumetric LR-spline vector fields and apply it to streamline rendering. However, the LR-spline approach is applicable to other visualization methods as well.

2.3. Multivariate Splines

Spline functions are piecewise polynomials that can be defined in any dimension. They can be used to compactly model scattered data, provided the trends are generally smooth, with local variations and few discontinuities. Splines have a variety of applications in scientific computing and are ubiquitous in fields such as computer aided design (CAD) where the ability to design freeform geometry is crucial. For a more detailed introduction to splines see [12]. Univariate splines have a straightforward generalization to multivariate tensor product splines, where each dimension is represented by a univariate spline. This is the classical method used to define spline surfaces and Non-Uniform Rational B-Splines (NURBS), the rational counterpart of splines which is widely used in CAD.

Despite its simplicity, the tensor-product generalization has some downsides when it comes to refinement. If a new knot value is inserted in one of the directions, the effect pervades throughout the entire domain in all other directions. The result is a global increase of the size of the representation, even when only local modelling flexibility is needed. This limits the value of tensor-product splines to relatively small examples, with few knots and low spatial dimension $n_d$ of the domain.

2.4. Adaptively Refined Splines

The increase of interest in splines in higher dimensions, driven by research efforts in fields such as isogeometric analysis, has resulted in several independent approaches to adaptive refinement of spline spaces in dimensions $n_d \geq 1$. These include hierarchical splines originally developed in [13] and extended in [14]; T-splines developed in [15]; and LR-splines developed in [16]. Though the details of the various methods differ, they all offer the ability to add degrees of freedom to the spline space locally in order to add modelling flexibility. They also share the property that the domain and the partitions of the domain are all axis-aligned. In addition, they can all be converted to tensor-product polynomials on individual elements.

Data: 3-dimensional scattered point data with associated field
Result: LR-spline approximating the data, achieved accuracy.
Initiate LR-spline data structure, with all coefficients $= 0$; Iterate approximation;
for levels = 1; levels < max levels; levels += 1 do
  if all points are within tolerance then exit;
  else
    Refine LR-spline volume;
    Iterate approximation;
  end
Cycle through to next refinement direction (i.e., contain $x, y$ or $z$);

Algorithm 1: Basic iterative approximation algorithm

3. Approximating Scattered Data with LR-splines

3.1. LR-splines background

For a detailed overview of the general theory of LR-splines see [16, 17]. In the context of this paper, LR-splines provide a compact representation for modelling generally smooth functions with detail that varies over multiple scales, since they support both local refinement and high degrees $\geq 1$. LR-splines are a superset of tensor-product splines and any LR-spline is generated by making local refinements from a tensor-product spline. The procedure for refinement is discussed in more detail in Section 3.2.2.

A vector-valued LR-spline function, $f : \mathbb{R}^3 \rightarrow \mathbb{R}^n$, is defined as a linear combination of functions known as LR B-splines:

$$f(x, y, z) = \sum_{i \in \Xi} c_i \gamma_i N^d_i(x, y, z),$$  

where $N^d_i$ are the LR B-splines of tri-degree $d = (d_1, d_2, d_3)$, $\gamma_i$ are scaling factors that ensure a partition of unity and $c_i \in \mathbb{R}^n$ are the coefficients. Each LR B-spline is locally equivalent to a tensor-product B-spline, but the knots that make up an LR B-spline are not necessarily strictly consecutive. The set of LR B-splines $N^d$ has no natural ordering, so we label them by an index set $\Xi$.

The domain of $f$ in $\mathbb{R}^3$ is a box partition, i.e., the partitioning of (e.g.) the interval $[0, 1]^3$ into a set of three-dimensional axis parallel boxes or elements. The two-dimensional boundaries of the elements are in the following called mesh-rectangles. The collection of all the mesh-rectangles in the box partition is called the LR-mesh. The domain, or support, of an individual LR B-spline is a minimal tensor mesh contained in the LR-mesh.

Refinements of an LR-spline is done by inserting a set of axis parallel mesh-rectangles in the LR-mesh, splitting several elements and the support of at least one LR B-spline. Multiple refinements can be inserted simultaneously, taking care to manage the splitting of the B-splines sequentially in order to ensure minimal support. An example visualizing the box structure of an LR spline is given in Fig. 1.
LR-splines have similar properties to tensor-product splines, such as minimal support basis functions and partition of unity. However, linear independence of the set of LR B-splines is not guaranteed and is a challenging problem for general refinements in 3-dimensions \([18]\). Linear independence is critical for applications in IGA, but for the purposes of visualization it does not pose a problem.

### 3.2. The LR-spline Approximation Algorithm

The motivation for approximating a data set using LR-splines is to achieve a representation of the input data that is compact whilst maintaining accuracy to within a given tolerance, and that is well-suited for interactive visualization. In our case the input data can be any scattered point data \((p_i)^{\infty}_{i=1}\), with associated scalar or vector values \((u_i)^{\infty}_{i=1}\). In the examples shown in Section 5 and Section 6, a number of data sources have been considered, including nodes of a tetrahedral finite-element mesh (telescope), a discrete element mesh (fluidized bed) and a regular voxel grid (MRI scan).

In this work, we have chosen a general iterative approximation algorithm, known as locally refined multi-level B-spline approximation, or LR-MBA. This approach puts few restrictions on the LR B-spline function set; in particular, there is no requirement for linear independence of the LR B-splines, even though the method is not a quasi-interpolant. An alternative approximation method is least-squares, though it requires linear independence of the LR B-splines, even though the method is not a quasi-interpolant. An alternative approximation method is least-squares, though it requires linear independence and we have therefore chosen not to explore it in this paper. By choosing an iterative approach, we can successively approximate and refine the LR-spline to ensure that the local resolution of the LR-mesh reflects the trends in the data, thereby minimising the degrees of freedom required by the approximation. By choosing tri-degree \(d = (2, 2, 2)\), the approximation is able to reproduce smooth trends in the data with relatively few refinements. The LR-MBA algorithm is based on the multilevel B-spline approximation (MBA) algorithm, first described in \([19]\). The approach was generalised to bivariate LR-splines in \([1]\) and in this paper we extend it to the case of volumetric LR-splines.

The basic algorithm is summarised in pseudocode in Algorithm 1 with the details described in the sections below. First the data structure is initialized as a simple LR-spline; typically, a Bézier block (i.e., a volumetric tensor-product of univariate Bézier intervals of respective degree \(d_1, d_2\) and \(d_3\)) with zero coefficients, although any LR-spline data structure can be used as the starting point for the approximation. The algorithm then proceeds to obtain an ever closer fit to the data by iteratively applying two main steps: refinement and approximation. The refinement step adds more degrees of freedom in regions where the error is too large and the approximation fits the LR-spline to the data. These steps are described in more detail below.

#### 3.2.1. Approximation Step

The approximation step is applied several times during the algorithm, first to generate an initial approximation and subsequently to iterate the approximation based on the applied refinements. The procedure updates the coefficients \(c_i\) of all LR B-splines \(N_i\) whose support contain points outside the prescribed tolerance. For each point \(p_i = (x_i, y_i, z_i)\), this is done by first assigning a quantity \(\phi_{jl}\) to each LR B-spline that contains the data point in its support:

\[
\phi_{jl} = \frac{\gamma_j N_j(x_i, y_i, z_i)}{\sum_l \gamma_l N_l(x_i, y_i, z_i)^2} \beta_l,
\]

where the index \(k\) runs over all such LR B-splines. Intuitively, the quantity \(\phi_{jl}\) is the minimum \(\ell^2\)-norm solution to the under-determined system of equations given by the problem of locally interpolating the point \(p_i\) with a vector-valued function \(f(x, y, z) = \sum_l \phi_l \gamma_l N_l(x_i, y_i, z_i)\). Since there will typically be several points in the support of each LR B-spline, there will also be multiple values of \(\phi_{jl}\) corresponding to each coefficient \(c_i\), and we must choose a unique value for \(c_i\) for the global representation. To do this we minimize the error

\[
e(c_i) = \sum_l |\gamma_l N_l(x_i, y_i, z_i) - \phi_{jl} \gamma_j N_j(x_i, y_i, z_i)|^2,
\]

where \(l\) runs through all values of \(l\) associated with points in the support of the current LR B-spline \(N_i\). This results in the following explicit formula for the new coefficients:

\[
c_i = \frac{\sum_l \gamma_l N_l(x_i, y_i, z_i)^2 \phi_{jl}}{\sum_l \gamma_l N_l(x_i, y_i, z_i)^2}.
\]

#### 3.2.2. Refinement Step

The refinement of the LR-spline data structure is divided into two parts; firstly, detecting which elements contain points that are not approximated within the tolerance, and secondly, inserting appropriate refinements to increase the degrees of freedom in the LR-spline.

![Fig. 1. Visualization of local box size with very coarse (left) and medium refinement (right); blue indicates large and red small box size.](image-url)
After an iteration of the approximation algorithm has been applied, there may be regions of the domain that only contain points that are sufficiently well approximated according to the prescribed tolerance. However, the elements that contain points outside the tolerance should be refined in order to add degrees of freedom to the approximating LR-spline space. The theory of LR-splines allows for great flexibility in the strategies that can be used to perform refinement [16]; however, there are some standard approaches. Since we require neither linear independence of the LR B-splines nor a well conditioned basis for the spline space, we have chosen a simple approach that inserts few refinements at each approximation level. The approach is known as minimum span refinement [17], and proceeds by only refining the LR B-spline with smallest support that covers an element with out-of-tolerance points. This approach ensures that at least one LR B-spline is always split and that the number of degrees of freedom are thereby increased. The refinement is applied in only one direction per refinement level, and we loop through the directions as the refinement level is increased.

In order to achieve a compact representation of the data, it is important to keep the number of refinements as small as possible, i.e., to use the available degrees of freedom at each refinement level as efficiently as possible. To do this, the approximation step must provide a good fit to the data before the error measurement is made to evaluate whether further refinements are necessary. Since the LR-MBA algorithm is an iterative approach that does not solve any global minimization problem, it can therefore be worth applying the several iterations of the approximation per refinement step. However, our experience is that by applying the approximation iteration more than once, we do not achieve significantly fewer refinements in the final approximation. Results on convergence of the LR-MBA algorithm will be the focus of future research.

4. Fast LR-Spline Evaluation

Locally refined LR-splines are well-suited for compact representations of volumetric data with overall smooth features and high local variation. Compared to uniformly refined splines, a main drawback of this flexibility is that the evaluation becomes more involved. This is due to the fact that LR-splines are a data format without a natural ordering of elements. To remedy this, we extend the approach presented in [3] to three dimensions.

Given a point in the parameter domain, one needs to first find the corresponding element $e_l$, see illustration in Fig. 2. For a large number $N$ of elements, a linear search through all elements, with its complexity of $O(N)$ in the average case, quickly becomes infeasible in the context of real-time rendering. Let the parameter domain be $[0, 1]^3$ for the following discussion. For quick element look-up

- **Texture.** For the special case where all knots in each direction can be written as multiples of $1/l$ where $l, i$ are natural numbers – typically $1/2^i$ – a look-up texture is the fastest method with a complexity of $O(1)$. However, it is apparent that this approach fails in many cases, e.g., if a knot value is an irrational number. In addition, the maximum size of OpenGL textures imposes a strict limit on the levels of local refinement.

- **Octree.** An octree can be used to remedy the restriction on the maximum level of refinement of the texture based approach. The search complexity of octrees is $O(\log(N))$. The downside of octrees with respect to LR-spline meshes is that they can only represent a regular data structure. In fact octrees only work if all knots can be written as multiples of $1/2^i$.

- **K-d Tree.** K-d trees are an excellent data structure for axis aligned splines with no restrictions on the knot values, i.e., for a general LR-spline. The average search complexity is $O(\log(N))$. However, since knot values can be at arbitrary parameter points, the split value needs to be stored and read from memory as well.

- **K-d Forest.** A k-d forest consists of a regular block division of the domain, where each block has its own k-d tree, see Fig. 2. This approach works for general LR-splines at arbitrary refinement levels (just as one k-d tree), but allows to exploit the efficiency of texture look-ups whenever possible. More details are presented in Section 4.1.

For all cases, the space requirement for the structure is $O(N)$. The overall approach consists of a k-d forest acceleration structure to identify the element containing a given parameter value, and a Bézier representation of the field in each element for efficient evaluation. A comparison of the different approaches is presented Section 5.5 and Section 6.3. These two components are described in the following sections.
tmp = access texture at value \( p \);
if (isLeaf = (32st bit of tmp == 1));
  elementNumber = first 29 bits of tmp;
  done;
else
  nextIndex = first 29 bits of tmp;
while true do
  knode = texelFetch(forest, nextIndex).rg;
  isLeaf = (32st bit bit of knode.r == 1);
  if (isLeaf then
    elementNumber = first 29 bits of knode.r;
    done;
  else
    dir (x, y, or z) = bits 30 and 31 of knode.r;
    if (p_dir <knode.g then
      nextIndex = first 29 bits of knode.r;
    else
      nextIndex += 1;
    end
  end
end

Algorithm 2: Algorithm (k-d forest) to obtain the element number, given a parameter value \( p \).

4.1. K-d Forest

In order to handle element look-up for general LR-splines efficiently, we present an approach that combines the speed of texture look-ups, with the necessary flexibility to handle general LR-splines provided by k-d trees. To this end, the parameter domain is divided into a set of \( I \times J \times K \) regular blocks \( \{ P_{i,j,k} \} \). For each of these blocks we build a k-d tree for all elements that intersect the block. Figure 2 schematically depicts the acceleration structure of the resulting k-d forest, i.e., multiple k-d trees. The influence of the number of trees on the evaluation time is discussed in Section 5 and Section 6.

The k-d forest is precomputed on the CPU and uploaded to the GPU. The head of each search tree is stored in a 3D texture for rapid look-ups in the search forest. This texture has the size \( I \times J \times K \) with internal format GL_R32UI. In the case of a single active element in a block \( P_{i,j,k} \) the element index is stored directly, otherwise it is interpreted as a pointer to the start of the search tree. The search forest is uploaded to a GL_RG32F texture buffer. The red component is used to store the indices/leafs, the green to store the split value. To minimize the amount of storage needed, we use the high bits to store if the node is a leaf or not, and if the split is in the \( x \)-, \( y \)-, or \( z \)-direction. A pseudo-algorithm for identifying the element containing a parameter value \( p \) is given in Algorithm 2.

In the worst case, i.e., for unbalanced trees, the search complexity is increased to \( O(N) \). When creating the search forest, it is therefore important to create depth-balanced trees, in order to ensure the search complexity of \( O(\log N) \). In order to achieve this, the correct direction to split, i.e., \( x \)-, \( y \)-, or \( z \)-direction, must be found. We employ an algorithm that is based on counting the active elements to the left and right of the mean split value in each direction.

4.2. Octree

In the special case, when all knots are can be written as multiples of \( 1/2^d \) (assuming a parameter domain \( [0, 1] \)), an octree can be used for efficient element look-up. The octree is pre-computed on the CPU and uploaded to the GPU as a GL_R32F texture buffer. The generation of the octree is straightforward; the algorithm starts with the whole parameter domain, then sub-divides the regions successively into 8 octree regions, whenever a region overlaps with more than one LR element. To minimize the amount of storage needed, we use a similar approach to the one presented for k-d tree/forest; the highest bit of each entry in the octree data structure can be used to indicate if the entry is a leaf or an element number.

4.3. Bézier blocks

The data structure of LR-splines is well-suited for efficient approximation and refinement, but comes at the expense of efficiency of evaluation. Once the approximation is completed, we therefore move to a representation better suited for efficient evaluation, consisting of a set of Bézier blocks. The transformation from LR B-splines to Bézier blocks is a change of basis, not an approximation, thus the representation quality is maintained, as well as the continuities between adjacent blocks. This basis shift is easily done by computing the control points of the interpolation polynomial. In addition to the coefficients, the value of the lower left and upper right corner of each element has to be stored on the GPU memory in a texture buffer. There are 4 main reasons for this change of representation:

- LR-splines are data structures without natural ordering, with coefficients distributed in memory. This is not optimal for an implementation on the GPU.
- The number of basis functions is the same for all blocks when evaluating in Bézier form, which is well-suited for evaluation on the GPU. In LR-spline form the number of basis functions can vary from element to element and is greater or equal to the number of basis functions needed to evaluate the same spline in Bézier form.
- LR-splines can not be evaluated in tensor product form: for a spline of degree \( d \), evaluating the same block in Bézier form needs \( 3d \) one-dimensional basis evaluations, while an LR-spline needs \( 3d^3 \).
- An added advantage of this conversion is that the algorithm and its implementation becomes independent of the representation; LR-splines, T-Splines, and hierarchical splines can all be converted to Bézier blocks and our algorithm applied for rendering.

With this strategy, well known algorithms, such as the De Casteljau’s algorithm, can readily be used to evaluate the volumetric splines on the GPU.
5. Volume Rendering

Classic volume rendering is a method to display a two-dimensional projection of a three-dimensional scalar field that is discretely sampled on a Cartesian grid. In order to achieve this, a model for radiative transfer is used to describe absorption and emission of light along view-rays. For a more detailed description of well-established techniques for volume rendering see [5], [20] and references therein. Here, we extend classic volume rendering to LR-splines.

5.1. Algorithm

Volume rendering is based on tracing view-rays through the volume from an imaginary observer. If such a view-ray intersects the object one obtains the color for the pixel of the screen by evaluating an integral describing the accumulated radiance along the ray. When taking both emission and absorption into account, the accumulated radiance $I$ along the ray is given by the so-called volume render integral

$$I(t) = I(0)T(s, t) + \int_{\gamma(s,t)} \sigma_{\lambda}(s)T_{\lambda}(s, t)ds,$$

where $\gamma$ denotes the line integral. The function $\sigma_{\lambda}(s)$ specifies emission, and $T_{\lambda}(s, t)$ specifies absorption (from $s$ to $t$) of light with the wavelength $\lambda$. In applications, one typically uses three groups of wavelengths representing red, green, and blue. The emission and absorption, defined by a so-called transfer function, depend on the value of the scalar field $\rho$.

Discretizations of Equation (2) are based on splitting the integral into intervals. Efficient implementations on GPUs are so-called compositing schemes where color and opacity is accumulated iteratively. Front-to-back compositing for the accumulated radiance $C_{\text{dst}} = (I, I_g, I_b)^T$, and the accumulated opacity $\alpha_{\text{dst}} = (1 - T_{\text{dst}})$ is given by Algorithm 3:

$$T \leftarrow (1 - \alpha_{\text{src}})^{\Delta s_i/\xi}$$
$$C_{\text{dst}} \leftarrow C_{\text{dst}} + (1 - T)(1 - \alpha_{\text{dst}})C_{\text{src}}$$
$$\alpha_{\text{dst}} \leftarrow \alpha_{\text{dst}} + (1 - T)(1 - \alpha_{\text{dst}})$$

Algorithm 3: Front-to-back compositing

Here, $\Delta s_i$ is the varying ray segment length and $\xi$ is a standard length. Furthermore, $C_{\text{src}}$ and $\alpha_{\text{src}}$ are given by the transfer function.

In many application areas transfer functions contain high frequency components, dictating a high sampling rate (Nyquist rate). One method is oversampling, i.e., introducing additional sampling points, although the underlying scalar field is approximately linear. Evaluating the scalar function in the setting of locally refined spline volume rendering is a time-intensive operation as it means evaluating Equation (1). To avoid oversampling, a common technique is pre-integration (see e.g., [21, 22]), which is based on calculating the volume render integral for pairs of sample values in advance. In this article we employ a technique called supersampling presented in [23], in order to account for the non-linearity of $I$. The approach is easy to implement, and successfully captures high frequencies of the transfer function. This increases the image quality by reducing so-called wood-grain artifacts, while avoiding computationally expensive evaluations of the LR-spline function.

5.2. Adaptive Sampling Distance

For LR-splines the size of the elements, e.g., as measured by diagonal length, can vary by several orders of magnitude. In the instance of the wind simulation presented in Fig. 5 the ratio between the largest and the smallest diagonal is 2048. A uniform sampling distance $\Delta s_i$ dictated by the smallest elements quickly becomes prohibitive for real time rendering. Because of the local refinement strategies of LR-splines, the highest level of refinements, i.e., the smallest elements, occur in areas with the largest local variation in the data. Thus the degree of variation of the field within each element can be expected to be of the same order of magnitude. As a result, an adaptive sampling $\Delta s_i$ in Algorithm 3 necessary for “capturing” all local details, is proportional to the local element size. We obtain an efficient algorithm by choosing the local step size to be a function of the length of the box sides at the point $p \in \mathbb{R}^3$, i.e.,

$$\Delta s_i(p) = \frac{1}{d^2} \min_{\text{box}(p)} \{\text{boxside}(p)\},$$

where $d$ is the degree of the spline in this element. In Table 1 we illustrate that this approach leads to a dramatic speed-up compared to a constant step size.
In order to enhance the perception of the data, gradient-based local illumination can be used. This lighting model assumes that external light is reflected at isosurfaces inside the volumetric data. Note that at each sample point both the scalar value and the gradient of the volume has to be evaluated. For an LR-spline or Bézier block, the gradient can be evaluated cheaply by reusing calculations for the scalar value. Applying this model to an STL object. To achieve the trimming, we employ a two-stage algorithm.

5.3. Local Volume Illumination

In order to enhance the perception of the data, gradient-based local illumination can be used. This lighting model assumes that external light is reflected at isosurfaces inside the volumetric data. Note that at each sample point both the scalar value and the gradient of the volume has to be evaluated. For an LR-spline or Bézier block, the gradient can be evaluated cheaply by reusing calculations for the scalar value. Applying this model to an STL object. To achieve the trimming, we employ a two-stage algorithm.

5.4. Trimming the Volume

An important tool for investigating a data set is to hide insignificant regions. This is particularly important when approximating simulation data on arbitrary geometries with LR-spline volumes that are axis-aligned. The values outside the valid geometry are artificial and need to be hidden. We trim the rectilinear geometry of the LR-spline volume by the boundary mesh of the original data. For a tetrahedral simulation mesh, this boundary mesh is a triangulated surface which can be represented by an STL object. To achieve the trimming, we employ a two-stage algorithm.

5.5. Results

In this section we provide several examples of data sets that are approximated by LR-spline volumes and analyze the rendering performance of the proposed algorithm. Screenshots for the case of the wind speed simulation around a telescope are presented in Fig. 5(a) and Fig. 5(b). The LR-volume has tri-degree (2,2,2) and consists of 36614 elements with 256 levels of refinement, i.e., the smallest box size is 256 times less than that of the largest. Since the simulation uses an STL file to define the boundary, and the LR-volume approximation naturally is an axis-aligned box, we need to trim. This trimming, i.e., the rendering of the front and back faces, takes about 2.2 ms for an STL comprising 2.4 million triangles. We have measured the

| case | Fig. 5(a) | Fig. 5(b) |
|------|-----------|-----------|
| fixed sampling distance [ms] | 504 | 102 |
| adaptive sampling distance [ms] | 96 | 36 |

Table 1. Comparison of rendering performance for fixed and adaptive sampling distance in the compositing scheme Algorithm 3 for the same visual result. The measurement was performed on an NVIDIA Titan GPU and a screen resolution of 1280 × 720.

| element search | texture | octree | k-d tree | k-d forest |
|----------------|---------|--------|----------|-----------|
| # trees^3      | 1       | 8      | 64       | 512       |
| min depth      | 12      | 0      | 0        | 0         |
| max depth /depth | 83 | 43     | 30       | 6         |
| mean depth     | 5.3     | 0.07   | 9.6e-5   | 0.02      |
| rendering [ms] | -       | 99     | 100      | 35        |

Table 2. Rendering times performed on an NVIDIA Titan GPU and a screen resolution of 1280 × 720.

1. Render the front and back faces of the STL object (which by definition has closed and watertight geometry) to a texture with twice the size of the screen. Enabling depth test ensures that the closest hits, i.e., front and back faces, are obtained.
2. Volume rendering is done from the front hit (or the near plane), to the back hit by a compositing scheme described above.

Fig. 4 shows examples for STL-trimming, cut-planes and clipping planes.

(c) MR-brain scan, see Fig. 5(c).
Fig. 5. Examples of visualization of scalar fields given by LR-spline volumes. The bars show the colors that are assigned to the values of the scalar field, where a checkerboard pattern indicates transparent regions.

performance (see Table 1) gain by the adaptive sampling distance described in Section 5.2 compared to the fixed sampling distance to be of the factor of 5 when rendering the whole scene and a factor of 3 for a camera position close to the telescope. Table 2(a) shows a comparison of rendering times with respect to the different element look-up strategies. Look-up textures are not applicable in this case, since the level of refinement is too high. Octree and k-d tree structures achieve approximately the same rendering time, where the octree has less depth but more children. K-d forest structure is the fastest element-search method. The table shows that the minimum depth, as well as the mean and variance of the trees quickly approaches zero with increasing number of trees in the k-d forest. This results in a decrease in the rendering times, as there are more and more cases where the element search can be performed by a single texture look-up instead of tree-traversal. However, even a texture size of $512^3$ is not sufficient to avoid the usage of k-d trees.

Fig. 5(c) shows a screen shot for the case of simulation of a fluidized bed. Particle velocity is given by an LR-spline of tri-degree (2,2,2) with 5311 elements and 13 refinement levels. In this case, neither textures, nor octrees are applicable, since there are knot values that cannot be written as multiples of $1/l$ where $l, i \in \mathbb{N}$. The rendering time improves with increasing number of trees in the forest, see Table 2(b). In addition, it can be deduced that local volume illumination increases the rendering time by approximately 80%, but allows to reveal important geometric information.

Finally, we present an MR brain scan given by an LR spline of tri-degree (2,2,2) with 255909 elements and 16 levels of refinement. Table 2(c) shows the rendering times for the different element-search algorithms. A look-up texture is the most efficient acceleration structure, but a k-d forest with $64^3$ trees is almost equally fast; the octree algorithm is the slowest.

6. Vector Field Rendering

Vector fields are typically used to represent physical properties such as a magnetic field or flow velocity. In the case of steady flow, i.e., a vector field that does not change with time, stream-, streak-, and pathlines coincide. For a given vector field $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^3$ a streamline with seed point $x_0 \in \mathbb{R}^3$ is given by the solution $x(t), t \in \mathbb{R}^+$ of the initial value problem

$$\dot{x} = f(x),$$
$$x(0) = x_0.$$  \hspace{1cm} (4)

In the following we will assume that $f$ is Lipschitz continuous, hence we have the existence of a unique local solution according to the theorem of Picard-Lindelöf. This assumption is fulfilled for LR-splines. For the rendering of streamlines we employ a two-stage algorithm.

1. Computation of streamlines: This stage is implemented as a compute shader and triggered only when seeds are set and/or changed, and/or a new field is loaded. Standard (higher order) discretization methods are applied and the result is stored in an RGBA32F image buffer of the size: number of seed points times maximum number of discrete sampling points.
2. Rendering of the computed streamlines: For each segment that is stored in the image buffer, a tube is rendered. For this tube a specific color map and lighting model is applied.

Although our description is limited to steady flow, an extension to unsteady flow involves an adaptation of the first stage only. Note, that it is possible to avoid having a fixed maximum number of discrete sample points per streamline by creating a linked list approach. We have chosen to use an image buffer with a fixed maximum number of discrete samples in order to achieve maximum performance.

6.1. Computation of Stream Lines

The computation of streamlines is done by discretizing the initial value problem \( \mathbf{x}_{n+1} = \mathbf{x}_n + h \sum_{i=1}^{s} b_i \mathbf{k}_i \), (5) with an explicit Runge Kutta (RK) method given by

\[
x_{n+1} = x_n + h \sum_{i=1}^{s} b_i k_i,
\]

where \( h \) is the step size, \( b_i \) are weights, and \( k_i = f(\cdot) \) are increments at locations certain locations specific to the RK method.

We would like to point the reader to [24] for a good introduction to numerical methods for ODEs. In this article we have implemented the following RK methods: (RK 1) explicit Euler method (first order), (RK 2) midpoint method (second order), (RK3) Kutta’s 3rd order method, (RK4) classic 4th order method, (RK4 3/8) 3/8 rule (4th order), (RKF) Runge-Kutta-Fehlberg method (5th order).

Note that in our case the evaluation of \( f \) means to evaluate an LR-spline. The RK method is terminated either if the streamline exits the LR-spline domain, or if the maximum number of samples is reached.

6.1.1. Adaptive Sampling Distance

In order to improve the efficiency of RK methods the step size \( h \) should be adaptive. In case of vector fields given by an LR-spline, \( h \) naturally depends on the relative size of the LR-elements and the degree of the underlying spline. To this end, we use the same local step size as described in Section 5.2, which leads to a speed-up of the computation, see Fig. 7.

6.2. Rendering of Tubes

The streamlines are visualized by rendering piecewise straight tubes, that are joined together by the discrete sample points \( x_n \) computed by the ODE solver in the compute stage. To
this end we draw \((\text{number of seed points}) \times (\text{maximum number of discrete sampling points})\) primitives (GL_POINTS) and run a program consisting of the following shader stages:

1. **Vertex shader**: The primitives are used to trigger rendering and the vertex shader sets a random position in order to be consistent with the OpenGL spec.

2. **Geometry shader**: The primitive ID (glPrimitiveIDIn) is used to determine the seed number \(s\), and sampling point \(n\). From this, the points \(x_n, x_{n+1}\) are extracted and used to compute and emit 4 points comprising the tube segment.

3. **Fragment shader**: Finally, the local position within an element is determined and a lighting model, e.g., Blinn-Phong, is applied. The intersections, start and end segments are treated such that they appear round. In addition, the tube can be colored for example by computing the local speed. This means evaluating the LR-spline at the local position and computing the Euclidean norm.

6.3. **Results**

In this section we provide examples showcasing interactive streamline computation and rendering. The computation of streamlines – although interactive computation for every frame is possible – is only necessary when the vector field or the seed points are initiated or changed.

The first two pictures in Fig. 6 show screenshots of rendering of the simulation results of wind flow around a telescope. The LR-spline with tri-degree \((2,2,2)\) has in total 88872 elements and 2028 refinement levels are used. Table 3(a) shows that all RK-methods achieve interactive frame rates for the computation of 88 streamlines. Regarding the different methods for element look-up, the texture based approach is not applicable due to size of the smallest element. The algorithm based on an octree is faster than a k-d forest based approach with 64\(^3\) trees. However, as the number of trees is increased, the k-d forest based approach becomes faster. For the views given in Fig. 6, the rendering of the streamlines takes about 7ms. Fig. 7 shows evidence that a regular sampling in this case is inferior to an LR-spline representation with respect to the number of degrees of freedom.

Finally, we present the case of particle velocity from a simulation of a fluidized bed. The LR-spline of tri-degree \((2,2,2)\) uses 5311 elements and 8 levels of refinement. Neither the texture based, nor the octree based approach are applicable for element search, since there are knot values that cannot be written as multiples of \(1/\ell_i\) where \(\ell, i \in \mathbb{N}\). Using the k-d forest based approach, the computation of 250 streamlines is performed with interactive frame rates, see Table 3(b). The streamline rendering with tubes takes about 7ms for the view presented in Fig. 6.

7. **Conclusion**

A novel, interactive method for volume visualization and streamline rendering for LR-splines has been presented. We have provide examples of data sets that are represented/approximated by volumetric LR-splines. Overall, the best acceleration structure for quick element look-up depends on the specific case. For examples with little refinement and restrictions on the knots values texture look-ups are fastest. In case of many levels of refinement and general knot placement a k-d forest acceleration structure becomes indispensable. Overall, interactive frame rates are demonstrated. In the future we plan to improve the convergence of the multilevel B-spline approximation (MBA) algorithm, and extend our algorithm to Isogeometric Analysis results and vector field rendering with LIC. Furthermore, we plan to analyze the relationship between the type of data and different ways to approximate it, such as regular sampling, octrees and LR-splines, with respect to the number of degrees of freedom as well as rendering performance.

**Acknowledgements**

The research leading to these results has received funding from the European Community’s Seventh Framework Pro-
gramme (FP7-ICT-2013-11) in the scope of the VELaSSCo project under Grant Agreement Number 619439. DEM data (Figure 1) courtesy of the School of Engineering at the University of Edinburgh. FEM data (Figure 2) courtesy the GiD and Kratos groups at CIMNE (the International Center for Numerical Methods in Engineering). ULG data (Figure 3) courtesy of the LOTAR project and Lockheed Martin.

References

[1] Skytt, V, Barrowclough, O, Dokken, T. Locally refined spline surfaces for representation of terrain data. Computers & Graphics 2015;49:58–68.

[2] Cottrell, JA, Hughes, TJ, Bazilevs, Y. Isogeometric analysis: toward integration of CAD and FEA. John Wiley & Sons; 2009.

[3] Hjelmervik, JM, Fuchs, FG. Interactive Pixel-Accurate Rendering of LR-Splines and T-Splines. In: Bickel, B, Ritschel, T, editors. EG 2015 - Short Papers. The Eurographics Association; 2015.

[4] Levoy, M. Display of surfaces from volume data. IEEE Computer Graphics & Applications 1988;8(3):29–37.

[5] Engel, K, Hadwiger, M, Kniss, JM, Rezk-Salama, C, Weiskopf, D. Real-time volume graphics. AK Peters; Limited; 2006.

[6] Kniss, J, McCormick, P, McPherson, A, Ahrens, J, Painter, J, Keahey, A, et al. Interactive texture-based volume rendering for large data sets. IEEE Computer Graphics and Applications 2001;21(4):52–61.

[7] Crassin, C, Neyret, F, Lefebvre, S, Eisemann, E. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In: Proceedings of the 2009 symposium on Interactive 3D graphics and games. ACM; 2009, p. 15–22.

[8] Kreylos, O, Weber, GH, Bethel, E, Shalf, JM, Hamann, B, Joy, KL. Remote interactive direct volume rendering of amr data. Lawrence Berkeley National Laboratory 2002.

[9] Marchesin, S, De Verdiere, GC. High-quality, semi-analytical volume rendering for amr data. IEEE transactions on visualization and computer graphics 2009;15(6):1611–1618.

[10] Beyer, J, Hadwiger, M, Pfister, H. A Survey of GPU-Based Large-Scale Volume Visualization. In: Borga, R, Maciejewski, R, Viola, L, editors. EuroVis - STARS. The Eurographics Association. ISBN 978-3-03868-028-4; 2014.

[11] Weiskopf, D, Hopf, M, Ertl, T. Hardware-accelerated visualization of time-varying 2d and 3d vector fields by texture advection via programmable per-pixel operations. In: In Vision, Modeling, and Visualization VMV 01 Conference Proceedings. 2001, p. 439–446.

[12] Farin, G. Curves and Surfaces for CAGD: A Practical Guide. 5th ed.; San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2002. ISBN 1-55860-737-4.

[13] Forsey, DR, Bartels, RH. Hierarchical B-spline refinement. In: ACM Siggraph Computer Graphics; vol. 22. ACM; 1988, p. 205–212.

[14] Giannelli, C, Jüttler, B, Speleers, H. THB-splines: The truncated basis for hierarchical splines. Computer Aided Geometric Design 2012;29(7):485–498.

[15] Sederberg, TW, Zheng, J, Bakenov, A, Nasri, A. T-splines and T-NURCCs. In: ACM transactions on graphics (TOG); vol. 22. ACM; 2003, p. 477–484.

[16] Dokken, T, Lyche, T, Pettersen, KF. Polynomial splines over locally refined box-partitions. Computer Aided Geometric Design 2013;30(3):331–356.

[17] Johannesen, KA, Kvamsdal, T, Dokken, T. Isogeometric analysis using LR B-splines. Computer Methods in Applied Mechanics and Engineering 2014;269:471–514.

[18] Pettersen, KF. On the dimension of multivariate spline spaces. SINTEF Report, A23875, ISBN 9788214053135 2013.

[19] Lee, S, Wolberg, G, Shin, SY. Scattered data interpolation with multilevel b-splines. IEEE transactions on visualization and computer graphics 1997;3(3):228–244.

[20] Jensen, HW. Realistic Image Synthesis Using Photon Mapping. A. K. Peters, Ltd.; 2001.

[21] Roettger, S, Guth, S, Weiskopf, D, Ertl, T, Strasser, W. Smart hardware-accelerated volume rendering. In: VisSym; vol. 3. Citeseer; 2003, p. 231–238.

[22] Engel, K, Kraus, M, Ertl, T. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware. HWWS ’01; New York, NY, USA: ACM. ISBN 1-58113-407-X; 2001, p. 9–16.

[23] Fuchs, F, Hjelmervik, J. Interactive isogeometric volume visualization with pixel-accurate geometry. Visualization and Computer Graphics, IEEE Transactions on 2016;22(2):1102–1114.

[24] Deuflhard, P, Bornemann. F. Scientific computing with ordinary differential equations; vol. 42 of Texts in Applied Mathematics. Springer; 2002. ISBN 9780387954622.