Real-Time Isosurface Extraction with View-Dependent Level of Detail and Applications

Manuel Scholz[1], Jan Bender[1] and Carsten Dachsbacher[2]

[1]Graduate School CE, TU Darmstadt [2]Karlsruhe Institute of Technology

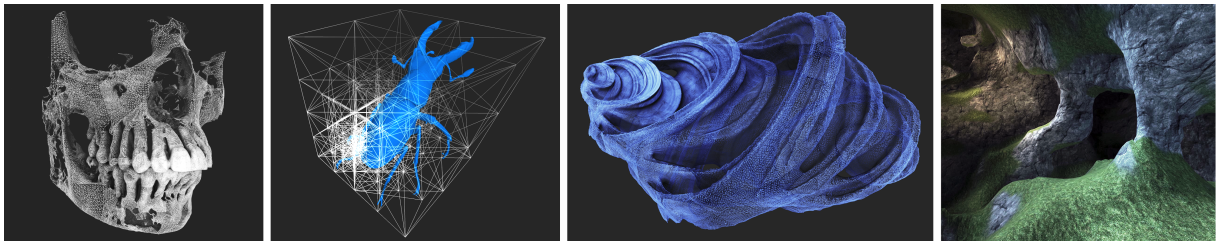


Figure 1: Isosurfaces rendered in real-time (above 200 frames per second) with our view-dependent level of detail algorithm. From left to right: a wireframe rendering of a medical dataset; a computer tomography dataset of a beetle with a resolution of $832 \times 832 \times 494$ (white lines illustrate our refinement hierarchy); a wireframe rendering of a Julia set fractal where the geometry is procedurally generated at run-time allowing for virtually unlimited detail; rendering of a terrain as an isosurface of complex procedurally-generated volume data.

Abstract

Volumetric scalar datasets are common in many scientific, engineering, and medical applications where they originate from measurements or simulations. Furthermore, they can represent geometric scene content, e.g. as distance or density fields. Often isosurfaces are extracted, either for indirect volume visualization in the former category, or to simply obtain a polygonal representation in case of the latter. However, even moderately sized volume datasets can result in complex isosurfaces which are challenging to recompute in real-time, e.g. when the user modifies the isovalue or when the data itself is dynamic. In this paper, we present a GPU-friendly algorithm for the extraction of isosurfaces, which provides adaptive level of detail rendering with view-dependent tessellation. It is based on a longest edge bisection scheme where the resulting tetrahedral cells are subdivided into four hexahedra, which then form the domain for the subsequent isosurface extraction step. Our algorithm generates meshes with good triangle quality even for highly nonlinear scalar data. In contrast to previous methods, it does not require any stitching between regions of different levels of detail. As all computation is performed at run-time and no preprocessing is required, the algorithm naturally supports dynamic data and allows us to change isovalues at any time.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

Volumetric scalar datasets are ubiquitous in scientific, technical, and medical applications. They originate directly from measurements, result from simulations, or are used to manually or procedurally create virtual scenes [PGGM09, RMD11]. In many cases, the volumetric scalar function is sampled and stored as grids. Obviously an efficient and interactive rendering of this data is crucial for many applications, including inspection and analysis, or simply for displaying the modelled content. One of the most popular visualization techniques for volume data is isosurface extraction which belongs to the so-called indirect volume visualization techniques. The basic techniques therefor, e.g. marching cubes [LC87], operate cell-by-cell on the grid, determine

the cells intersecting the isosurface, and create a piecewise polygonal approximation to the isosurface. However, even for moderately sized datasets, isosurfaces can get very complex which makes their interactive visualization a challenging problem. On the other hand, in most cases not all parts of an isosurface need to be extracted with the same accuracy or detail, e.g. as their projected image space size varies and they are simply less important to the viewer. At this point, level of detail (LOD) techniques can be used to reduce the memory footprint as well as the required computational resources without noticeably sacrificing quality.

Existing visualization methods for isosurface extraction from large and complex datasets require either expensive and time consuming preprocessing of the data, or yield low frame rates even on modern graphics hardware. Obviously, this contradicts the requirements of many applications where instant inspection of measurements or simulation results, with possibly time-varying data, is required. Preprocessing is also not applicable whenever the volume data is procedurally created or depending on user-input. This aspect is of particular importance in this paper, as one of our applications is the rendering of terrains defined by procedurally generated scalar data.

The contribution of this paper is twofold: First, we propose a novel LOD algorithm for isosurface rendering on modern graphics hardware which enables interactive indirect volume visualization of large data sets. By performing incremental updates to our LOD hierarchy and amortizing computational cost over several frames, high display rates are achieved. Unlike previous methods, which first extract a full resolution base mesh and build a LOD hierarchy in a bottom-up manner, our method does not rely on any preprocessing. As a consequence, our algorithm can update the complete isosurface at run-time within a few seconds which allows for adjusting the isovalue during exploration. We use hexahedral cells to extract the isosurface from the volumetric representation. This results in more a desirable topology, a better triangle quality and less primitives than most other multiresolution meshing techniques which rely on tetrahedral representations. Our algorithm also avoids costly stitching processes that are required with most previous approaches. Finally, we discuss aliasing in the context of level of detail rendering and present a sampling strategy applicable to many volumetric representations. Four examples of complex isosurfaces rendered with our method are shown in Figure 1.

As a second contribution, we further elaborate on the use of our isosurface rendering method for terrain visualization. This is a challenging example for demonstrating its application to procedurally, potentially time-varying, and virtually infinite content generated on-the-fly, which makes any precomputation impossible. Terrain visualization has a long history in computer graphics and is a key component of many virtual reality applications, simulators, geographic information systems, and computer games. Most research in this field has focused on heightmap-based approaches which facilitate effective level of detail strategies for rendering terrain at large scales. However, they are unable to faithfully represent features like caves, overhangs, rugged mountains and steep cliffs. As the expectations of users regarding detail and realism increase, this limitation becomes more apparent. Often these missing features are added by placing separate objects on top of the terrain, which can only be considered as a work-around rather than a satisfactory solution to the problem. Consequently, volumetric approaches have recently gained more attention, e.g. in the modelling system of Peytavie et al. [PGGM09]. These can represent arbitrary shapes and thus provide more flexibility than heightmaps. In this setting, a terrain surface is typically defined by a scalar function from which an isosurface is extracted. Our LOD algorithm delivers a tool for rendering these isosurfaces at frame rates comparable to heightmap-based approaches.

2. Related Work

Analogous to the twofold contribution of our paper, we first discuss previous work related to isosurface extraction in general, followed by a brief overview over terrain rendering methods.

2.1. Isosurface Extraction

Basic Techniques The basic techniques for isosurface extraction operate cell-by-cell on scalar functions stored on the vertices of a static grid. The marching cubes algorithm [LC87] is probably the most well-known isosurfacing algorithm. It assumes cubical lattices, but the core idea (the cell-by-cell operation) has quickly been generalized to other types of grids, e.g. to curved or irregular grids. Early popular works for the latter include the marching tetrahedra algorithm [PT90, ST90]. Methods that process a grid cell-wise obviously perform badly on large grids, as every cell has to be tested whether it intersects the isosurface. One strategy to reduce this cost are data structures to query the required cells for a given isovalue. Cignoni et al. [CMPS96], for example, use interval trees built on the minimum and maximum scalar values of each cell to determine all cells intersecting the isovalue in optimal time. Note that the isosurface extraction is still non-adaptive, as its precision is coupled to the grid resolution. For very large data sets, however, it is desirable to adapt the tessellation to the image space projection of the surface.

Adaptive Tessellation In order to adapt the resolution of an isosurface, multiresolution methods have been introduced, e.g. for structured rectilinear grids [WKL*01, WKE99, WCM12] or tetrahedral grids [GR99, Ger02]. Note that many of these approaches require stitching to connect regions of the isosurface with varying degree of tessellation or operate at a fine-grained level making them impractical for large datasets. Dual contouring [JLSW02] is probably

one of the most popular adaptive algorithms. It extracts isosurfaces with varying resolutions using an octree representation of the scalar data. Each octree cell contains a single vertex which is moved to the isosurface by quadric error minimization. Then all edges of the octree cells are inspected for a sign change of the isofunction and vertices of cells adjacent to the respective edge are connected to form a part of the isosurface. For large isosurfaces dual contouring becomes inefficient because it operates at the triangle level and imposes a large overhead for octree management and traversal. Furthermore, in certain configurations the resulting triangulation can contain undesirable foldovers and artifacts.

Most related to our algorithm are the following two works which do not use octrees and avoid expensive stitching operations. Gregorski et al. [GDL*02] build a conforming tetrahedral mesh with an adaptive longest edge bisection hierarchy to extract the isosurface. Pascucci et al. [Pas04] improve this method by transferring the surface extraction phase to the GPU. Compared to our method, extracting a surface directly from adaptive tetrahedral meshes results in a larger hierarchy overhead, significantly more triangles and a lower mesh quality (see Figure 6). Additionally our algorithm does not suffer from certain artifacts that arise from contouring non-linear isofunctions on simplex grids.

For a more comprehensive overview, we refer the reader to Newman and Yi's excellent survey [NY06] over the evolution of marching cubes and its successors.

Ray casting Isosurfaces Isosurfaces can also be rendered without explicitly generating a polygonal mesh by finding the intersection of view rays with the isosurface, e.g. using ray marching. These approaches provide inherent level of detail (e.g. by adapting step sizes in ray marching) and typically deliver good image quality. Their disadvantage is the lower rendering performance which, however, is somewhat mitigated by the increasing power of modern GPUs. Today there exist many ray casting algorithms that reach interactive frame rates, e.g. [YLM06, GMG08, CNLE09, Áfr12]. However, it is worth noting that the performance of ray casting is output-sensitive, i.e. their rendering performance does not scale well to high display resolutions; on the other hand, rasterization based methods can offer better performance and allow for trading quality for speed more easily. To combine the best from both worlds, hybrid approaches were developed which dynamically switch from ray casting to rasterization to optimize performance [ZQHK04, GM05]. Note that these methods depend on acceleration data structures which are built in a preprocessing step and thus do not meet one important goal of our approach. For example, Reichl et al. [RCBW12] proposed a hybrid approach which uses a sample based surface representation which makes it possible to outperforms rasterization as well as ray tracing seen individually. Their algorithm uses triangle meshes or regular volume datasets as input and requires only little preprocessing time. However, dynamically building these data structures at full resolution can consume significant amounts of computational resources and quickly become impractical. Also, when rendering isosurface no explicit triangulation is build from the volume dataset. Hence, their hybrid approach falls back to classical isosurface raytracing instead of using efficient rasterization in closeup regions.

2.2. Terrain Visualization

Methods for rendering large scale terrain data can be roughly categorized into heightmap-based methods, (generic) massive model visualization techniques, and volumetric terrain rendering. For further reading, we refer to surveys on (multiresolution) terrain models [Dac06, PG07] for the "classic" techniques, and to [Vir14] which provides an extensive collection of references.

Heightmap-Based Terrain Rendering Due to its simplicity and compactness, terrains have traditionally been stored as heightmaps. Previous work on rendering from this representation spans the entire range from fine-grained level of detail methods that operate on the individual triangles, e.g. [LKR*96, DWS*97], to coarse grained methods which became the first choice with the advent of powerful graphics hardware. The approaches tailored for modern GPUs adjust the detail level on the granularity of large chunks of geometry and tessellate their borders to match the resolution of neighbor chunks [LH04, Str09, BGP09, LKES09, JL12]. In the same spirit as for isosurfaces, hybrid approaches combining rasterization and ray casting have been proposed [DKW09, DKW10, AGD10] to cope with the strongly varying detail with terrain rendering. Heightmaps are well suited to model the shape of natural terrain at large scales, but apparently lack the ability to represent caves or overhangs, and provide low sampling at steep slopes.

Massive Model Visualization Techniques Techniques that are designed to render arbitrary highly detailed models can obviously also be applied to rendering terrains. In contrast to heightmaps, these techniques do not impose any restrictions on the topology and shape of the terrain. Several techniques exist to render meshes of up to several hundred million polygons at interactive rates. Cignoni et al. [CGG*03] and Lario et al. [LPT03] present methods for rendering terrain with triangulated irregular networks (TINs). Although only demonstrated for heightmap datasets in these works, TINs can also represent overhangs and other complex terrain features. For LOD rendering the geometry is often stored in a tree data structure where each node contains a part of the geometry at a certain level of detail [CGG*04, SM05, BGB*05]. The tree is constructed by recursively simplifying and merging the geometry of nodes. For rendering, a front tracking approach is used to select the appropriate nodes for the current view. Cignoni et al. [GMC*06] extend these methods by introducing a compression scheme for the large geometry

chunks. Gobbetti and Marton [GM05] propose to complement the polygonal rendering by switching to a precomputed voxel representation of the geometry if the screen space footprint of a node becomes smaller than a few pixels. Hu et al. [HSH09] extend the idea of progressive meshes [Hop96] with a view-dependent refinement scheme which, however, requires a significant amount of GPU resources for hierarchy management.

Despite their ability to render large and complex terrains, massive model visualization techniques require an explicit representation of the mesh which consumes a large amount of memory. More importantly, all of the previously mentioned methods rely on an expensive preprocessing step to build the required data structures. This makes these methods inapplicable for online generated procedural content.

Volumetric Terrain In recent years other representations (than polygonal meshes) for rendering the terrain surface gained more attention. Peytavie et al. [PGGM09] present a modeling framework which uses a compact stack-based representation, somewhat similar to a run-length compression. They demonstrate several modeling tools for their terrain representation, but do not present a level of detail algorithm. Loeffler et al. [LMS11] introduce a real-time rendering technique for stack-based terrains. They transform this representation into an octree data structure for rendering. Their method requires a stitching process for regions where cells of different resolution meet. Stitching has to be performed whenever the neighborhood of a cell changes. The need for frequent recomputation of cell geometries prevents efficient caching of the extracted isosurface and stresses computational resources. In our algorithm, cells are treated independently and no stitching is required, which avoids these problems. Note that stack-based terrain representations can also be used with our LOD algorithm.

Obviously, there are methods spanning more than one category in our classification of prior work. Gobbetti and Marton [GM05], for example, propose hybrid polygon-volumetric representations. In particular, voxel-based representations have been studied intensely in recent years, as modern GPUs provide increasing memory and are very efficient with ray marching [Len10, Cra12].

3. Level Of Detail Algorithm

The main goal of our methods is to render isosurfaces from volumetric scalar functions. These are approximated by a triangle mesh for efficient rendering on graphics hardware.

To be able to explore large and complex isosurfaces interactively, a level of detail (LOD) algorithm is necessary to reduce rendering and storage cost. LOD algorithms are usually based on a hierarchical decomposition of the domain space. Previous approaches which do not rely on precomputed mesh representations use either a very fine-grained tetrahedral subdivision, or employ an octree partitioning and generate cubical cells with varying size. In each cube a regular lattice of fixed resolution is placed which is then used to extract a triangular approximation of the isosurface. A problem with this approach is that cracks in the extracted surface can occur where cells of different resolution meet. These cracks must be fixed by a subsequent stitching process which increases the system complexity and degrades performance. Fine-grained methods on the other hand generate only a few triangles per cell which leads to very large hierarchies. These tend to consume a significant amount of computational and storage resources. Our method avoids these issues by using a coarse grained conforming hexahedral subdivision where each cell contains a regular lattice of fixed resolution. By definition neighboring cells in such a subdivision always coincide so that the regular lattices align perfectly at cell boundaries. Therefore, the triangle mesh extracted from the lattices exhibits no cracks in its triangulation (see Figure 2).

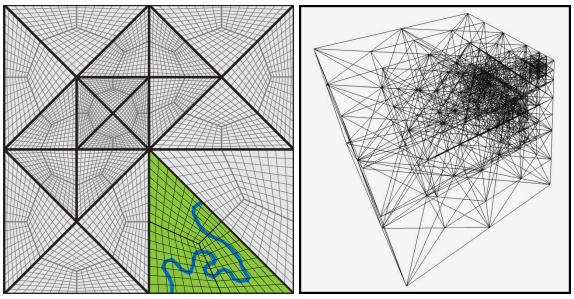


Figure 2: Left: The concept of our LOD algorithm in 2D: Thick black lines show the longest edge bisection (LEB) hierarchy. The lattices used for surface extraction are depicted in gray. Note that the lattices match at cell borders so that no stitching is required. Inside the green cell a part of the surface is shown in blue. Right: A longest edge bisection tetrahedra hierarchy in 3D.

To allow an interactive navigation or fly-through of the isosurface, the hexahedral mesh has to adapt quickly to new viewer positions. We meet this requirement in two steps: First, a diamond hierarchy [WDF08] is used to efficiently build an adaptive conforming tetrahedral partitioning

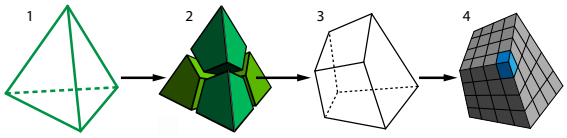
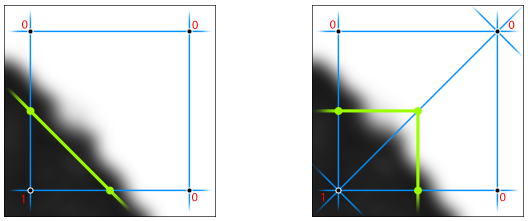
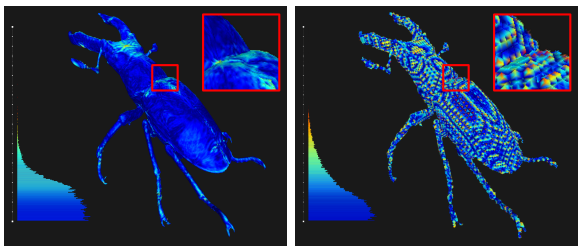


Figure 3: A tetrahedral cell (1) in the LEB hierarchy is split into four hexahedra (2). Each of the hexahedra (3) is regularly subdivided into a fixed resolution lattice (4) of hexahedral elements (blue). These are used to extract the surface using the marching cubes algorithm.



(a) (b)

Figure 4: Surface approximation in cubical cells (a) and simplex cells (b) in 2D. Cell edges are depicted in blue, the scalar function is shown as a black-white gradient and sampled values are shown in red. The edge intersection points, retrieved by linear root finding, are drawn in green. Note how right angled artifacts can arise on simplex grids when the scalar function is strongly nonlinear.

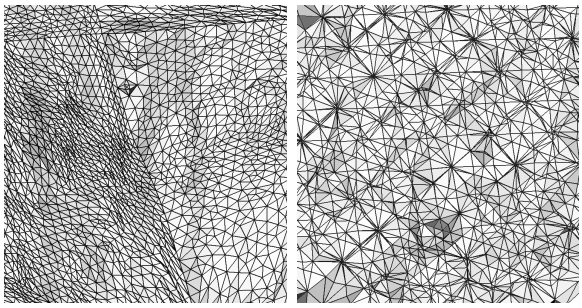


(a) (b)

Figure 5: Geometric error on the stag beetle dataset with uniform level of detail. Blue denotes low error with respect to ground truth, red denotes large errors. (a) Our method using hexahedral elements. (b) The method of Gregorski et al. [GDL*02] using the tetrahedral elements of the diamond hierarchy directly for surface extraction. Both models use a comparable number of triangles and vertices. Note the strong interpolation artifacts in (b) caused by nonlinearities in the scalar function.

of space. Second, each tetrahedron is split into four hexahedra (see Figure 3) before the triangle surface is extracted with a modified marching cubes (MC) algorithm. The subdivision of tetrahedral cells into hexahedra and the surface extraction with the MC algorithm is motivated as follows: MC is known to create better isosurface approximations with less artifacts compared to simplex-based methods [CMS06]. These artifacts occur in highly nonlinear regions of the scalar functions as can be seen in Figures 4 and 5. MC generated meshes also have a lower number of triangles and better mesh topology with a more evenly distributed vertex valence (see Figure 6). Note that these properties persist even though we use deformed hexahedral cells instead of cubic cells.

For the surface extraction step we propose a novel sampling strategy which maintains the independence of cells and



(a) (b)

Figure 6: (a) The mesh structure of a flat region generated by our method. (b) Isosurface extracted directly from the tetrahedra of the diamond hierarchy [GDL*02].

avoids aliasing. It is based on a frequency space decomposition of the scalar function and employs a special interpolation scheme which controls an adaptive low-pass filter to suppress aliasing artifacts.

In the following, we detail the three major parts of our method. In Section 3.1 we describe the level of detail hierarchy. Section 3.2 deals with the extraction of the isosurface inside a single cell of the hierarchy. Finally, Section 3.3 presents the sampling strategy that supresses aliasing while maintaining cell independency. Figure 7 shows an overview of the individual stages of the algorithm. In our implementation, all three components are executed on the CPU. Our results indicate that our method fully utilizes the GPU and thus makes a balanced use of computational resources.

3.1. Level of Detail Hierarchy

The level of detail hierarchy is a data structure responsible for adapting the resolution of the triangle mesh to the current viewing conditions. Our LOD algorithm uses three-dimensional diamond hierarchies [WF11] which are based on the longest edge bisection (LEB) of tetrahedra. We start with a cubic domain that is initially split into six tetrahedra, which are then subdivided according to the LEB scheme as needed. An interesting property is that tetrahedra created by the LEB scheme can be divided into three congruence classes. Recursively splitting a tetrahedron three times yields smaller tetrahedra that have exactly the same shape and quality as the initial one. Therefore, element quality does not degrade during subdivision and it is guaranteed that only well-shaped tetrahedra are created. This is crucial for obtaining triangle meshes of good quality.

The hierarchy is stored as a binary tree where each node represents a single tetrahedron. Tetrahedra that share a common longest edge form a diamond and must be split simultaneously to maintain a conforming tetrahedralization. We use the encoding scheme by Weiss and De Floriani [WDF08]

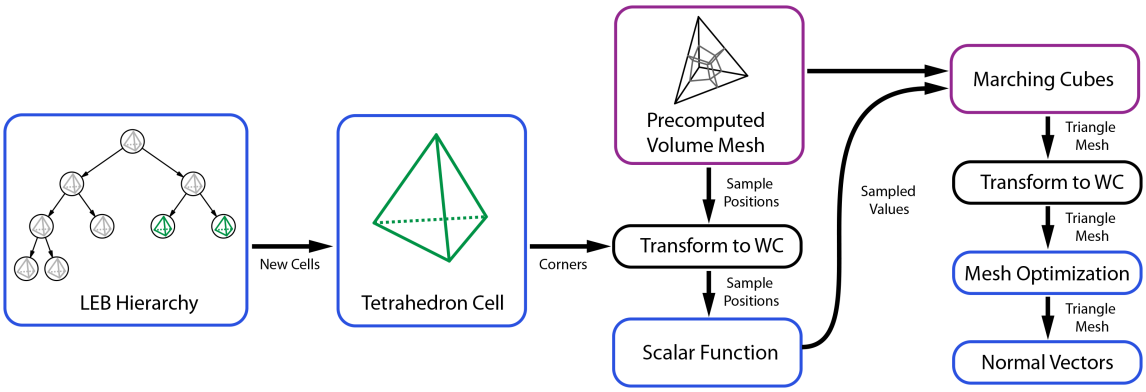


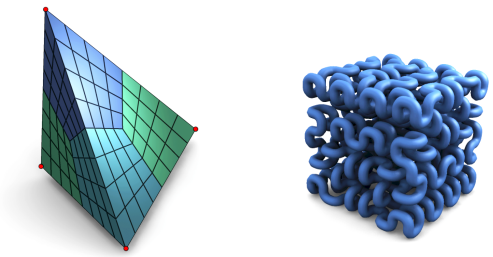
Figure 7: Algorithm overview: Blue frames denote steps that operate in Cartesian world coordinates (WC), while steps with purple frames operate in barycentric coordinates. When the hierarchy is refined, new cells are passed to the surface extraction step. Its corner positions are used to transform sample and vertex positions from barycentric coordinates into world space.

to access diamonds and check for dependencies efficiently. For a fast hierarchy traversal, diamonds are indexed in a hashmap by their central vertex position. The leaf nodes of the binary tree form the active front and define the current conforming tetrahedral mesh. The active front is refined and coarsened in real-time to adapt the mesh resolution to the viewer position. Whenever a node of the active front is split, the surface extraction procedure is invoked for both of its children. This process is subject of the next subsection.

As stated above, our algorithm supports arbitrary refinement criteria. Our prototype implementation uses a simple distance-based method which works well for most applications. For each node the distance l of its bounding sphere to the viewer is computed. The target refinement depth d_t is then defined as:

$$d_t = \log_a(l). \tag{1}$$

A node is split whenever its target depth d_t is larger than its current depth d_h inside the LEB hierarchy. The parameter a controls the relationship between detail level and distance, and should be chosen according to the field of view of the camera. We also add a small hysteresis of at least one hierarchy level to control the collapsing of nodes.



(a) (b)

Figure 8: (a) The volume mesh contains all four lattices with $4 \times 4 \times 4$ hexahedral elements each. Its vertex positions are stored in four-dimensional barycentric coordinates. (b) We use a three-dimensional Hilbert curve to reorder hexahedral elements for improved locality inside the hexahedra of the volume mesh.

3.2. Surface Extraction

When a cell in the level of detail hierarchy is created, the surface inside this new cell has to be extracted from the scalar function. Since each cell is part of a conforming tetrahedral mesh, no stitching and adaption to its neighbors is required. Furthermore, cells are independent from each other and the triangulation of the enclosed surface does not have to be changed once it has been created. This property is one of the main advantages over previous methods and has the following benefits:

- The triangle mesh of a cell does not have to be adapted to its neighboring cells. No computation time is required for stitching, which makes the system very efficient.
- A cell's triangle mesh can be cached for later reuse.
- Since the geometry of a cell remains static, updates of GPU rendering buffers are required less frequently than in previous methods.

The surface extraction is a performance critical operation usually done at run-time. Our goal is to extract an isosurface inside a single tetrahedral cell in form of an indexed face set which can directly be used for rendering. For this task we extend the original MC algorithm to operate on arbitrary hexahedral *volume meshes* instead of regularly subdivided cubic domains. In our work, the volume mesh has the shape

of a tetrahedron containing four hexahedral lattices (see Figure 8a). Since all cells of the LOD hierarchy are subdivided in the same way, we can reuse a single volume mesh for all cells. This requires a representation that is independent from the actual shape, position and rotation of a cell. We therefore store the volume mesh in four-dimensional barycentric coordinates and transform it to world coordinates during surface extraction when the cell geometry is available. The volume mesh is precomputed at startup in three steps: First a tetrahedron is split into four hexahedra H_i. Each of them is further subdivided into a regular lattice of smaller hexahedral elements. The elements of all four hexahedra H_i are then unified into a single volume mesh representation (see Figure 8a), which is stored as lists of vertices, edges and hexahedral elements E_j (see Figure 3). To increase the locality of subsequent elements E_j, we reorder the elements of each hexahedron H_i based on a three-dimensional Hilbert curve (see Figure 8b).

Later, the MC algorithm uses the same element order to extract the isosurface. This results in a more spatially coherent triangle mesh layout which improves vertex cache utilization during rendering.

The modified MC algorithm can be described in three steps: First, all vertex positions of the volume mesh are transformed to world space to evaluate the scalar function. Second, each edge of the volume mesh is inspected. If a sign change in the previously sampled values across an edge is detected, a new triangle vertex is added to the indexed face set data structure. The index of this new vertex is stored in the respective edge of the volume mesh. Its position is computed by linear root finding as in the original MC algorithm. Third, every cell of the volume mesh is traversed and the triangulation of the isosurface is fetched from the MC lookup table. The final vertex indices of each triangle are resolved using the index values which were previously stored in the volume mesh edges. The indices of each new triangle are then added to the indexed face set. In the last step, the triangle mesh is transformed from barycentric coordinates into worldspace using the corner positions of its associated tetrahedral cell.

As the MC algorithm tends to produce skinny triangles, we apply Laplacian smoothing as a post processing step. Optionally, we perform multiple choice quadric error mesh simplification [WK02] to reduce the total number of triangles while preserving a good approximation of the original geometry. Both algorithms are implemented to work directly on indexed face sets.

Vertex normal vectors of the final mesh are computed from the gradient of the scalar function by a central differences approximation along the world coordinate system axis. This process requires six additional samples per surface vertex but yields consistent normal vectors at cell boundaries without breaking the independence of cells. For scalar functions which are very expensive to evaluate the normal vectors can also be computed using only four samples by choosing a different numerical differentiation technique.

3.3. Sampling

The shape of an isosurface is defined by its scalar function which needs to be sampled at discrete positions in order to extract a triangle mesh. Our LOD algorithm does not impose any restrictions on this function. Nevertheless, discrete sampling often leads to aliasing artifacts which degrade visual quality and make LOD transitions more noticeable. To mitigate this problem we apply a spatially-varying low-pass filter before sampling is performed. A naive implementation as a discrete filter is very general, but would require prohibitively many evaluations of the scalar function during run-time.

To accelerate filtering, we express scalar functions θ as a sum of terms $d_i\colon \mathbb{R}^3 \to \mathbb{R}$ that depend on the position $\mathbf{p}$ and have a small extend in frequency space. Such a representation can be found for many procedural volumetric models and obtained for discrete volume data. For example, discrete volume data can be converted into a Laplace pyramid, where each pyramid level provides one term d_i. High frequencies are removed by weighting each d_i according to its dominant frequency f_i and the filter radius r:

$$\theta(\mathbf{p}, r) = \sum_{i=0}^{n} d_i(\mathbf{p}) w(f_i, r). \tag{2}$$

The weighting function w is defined as the frequency transformation of a Gauss filter kernel:

$$w(f_i, r) = e^{-f_i^2 r^2}. \tag{3}$$

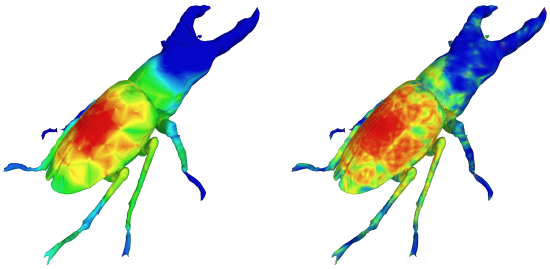
To guarantee the consistency of sampled values across cell borders the filter radius r must be derived only from information that is available to all cells adjacent to a certain sample location. Therefore, we base its computation on the length of the cells' edges. For a sample located on an edge, r is computed by dividing the edge length by the grid size of the volume mesh. For samples that are not located on an edge we interpolate the sample radii $\mathbf{s} \in \mathbb{R}^6$ of the six edges of the cell as follows:

$$r(\mathbf{x}) = \mathbf{s} \cdot w(\mathbf{x}) \tag{4}$$

with $\mathbf{x} \in \mathbb{R}^4$ being the barycentric coordinates of the sample location and

$$w(\mathbf{x}) = \frac{1}{\sum_{i=1}^{6} \overline{w}_i(\mathbf{x})} \overline{w}, \qquad \overline{w}(\mathbf{x}) = \begin{pmatrix} \mathbf{x}_1\mathbf{x}_2 \\ \mathbf{x}_1\mathbf{x}_3 \\ \mathbf{x}_1\mathbf{x}_4 \\ \mathbf{x}_2\mathbf{x}_3 \\ \mathbf{x}_2\mathbf{x}_4 \\ \mathbf{x}_3\mathbf{x}_4 \end{pmatrix}. \tag{5}$$

Note that the weights w can be precomputed for each vertex of the volume mesh. For samples located at cell corners, the filter radius is undefined as adjacent cells share only the



(a) (b)

Figure 9: (a) Color coded radius of the adaptive low-pass filter kernel which is used to suppress aliasing. The filter radius function is continuous across the entire domain (except for cell corners) so that coherent filtering results across cell borders are achieved. (b) Shows the sizes of the triangles for the extracted mesh for comparison.

information about the respective corner position. This is reflected by the singularities of the interpolant at cell corners. For these samples we set the filter radius to the same value as for samples of the most detailed LOD level. This yields correct results close to the observer and produces small errors for lower detail levels that are hardly noticeable in practice. Figure 9 shows the color coded filter radius on the surface of the extracted mesh and its correlation to the triangle size.

4. Rendering

In this section we explain how to efficiently render the geometry generated in the surface extraction phase. When the viewer explores the isosurface, the active front is constantly modified (cf. Section 3.1). The number of triangles and vertices generated during a refinement operation may vary strongly depending on the surface area of the isosurface inside the cells. This is challenging as GPU resource allocation during run-time is typically slow and drawing a large number of small triangle meshes is inefficient on graphics hardware.

To overcome these problems we introduce an additional front into the LEB hierarchy denoted as the *GPU buffer front* (see Figure 10). An entry of the GPU-buffer front contains the geometry of several active front nodes to create larger chunks of fixed sized data. If the size of this geometry exceeds the capacity of an GPU-buffer front entry, it is split into two equal sized entries and the front moves downwards. When the geometry of two sibling GPU-buffer entries fits into a single buffer, the two entries are merged together and the GPU-buffer front moves upwards. This approach has two advantages: First, rendering is more efficient because the triangle batch size is increased while the graphics API draw call count is decreased. Second, the vertex and index buffers

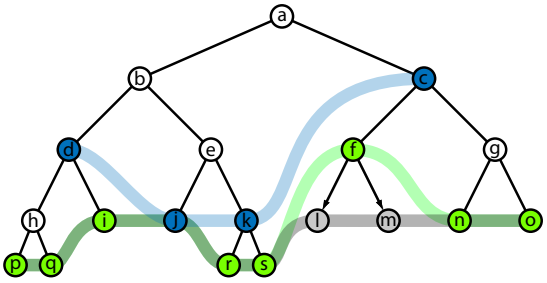


Figure 10: Visualization of the active front (green) and the GPU-buffer front (blue). Hierarchy nodes are represented as circles. The gray line depicts the active front after node f is split.

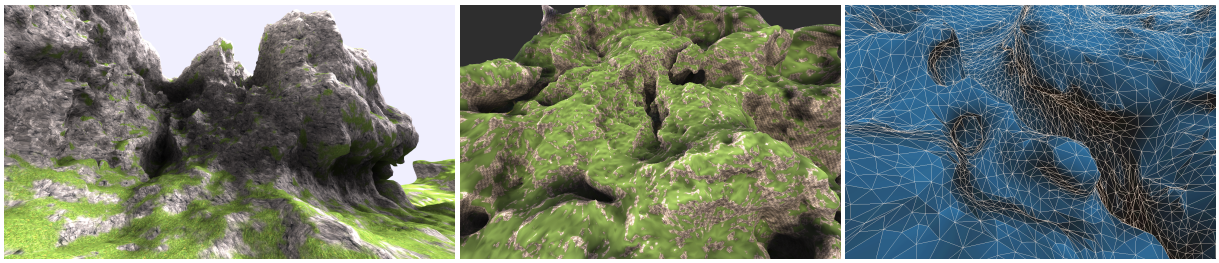
are all of the same size and can therefore be reused easily; this avoids costly GPU resource allocation at run-time.

During rendering, all entries of the GPU-buffer front are traversed and their associated geometry is drawn. The surface extraction process and the previously mentioned merging of triangle meshes is performed in a separate background thread which allows the main rendering loop to operate at high frame rates.

Many massive model visualization techniques use geomorphing to make LOD transitions less visible: They smoothly move the vertices of a high resolution mesh onto the surface of its low resolution counterpart before replacing it completely. This requires a correspondence between surfaces of different detail levels. Since we use a top down approach without preprocessing, the topology of the extracted surface may change during a LOD transition. In general a direct correspondence between surfaces of different detail levels can therefore not be found so that geomorphing cannot be implemented with our system. However, image space blending as discussed by Giegl and Wimmer [GW07] provides a viable alternative that is commonly used in industry and has shown to work well in practice.

5. Results

In this section we present the results of our method and its application to terrain rendering and indirect volume visualization. We implemented our algorithm in C++ and used the DirectX 11 API for rendering. All tests were carried out on a quad core AMD Phenom II 3.2 GHz processor and an NVIDIA GeForce GTX 580 graphics card. As mentioned earlier, in our implementation the surface extraction as well as the hierarchy updates are performed in a background thread on the CPU to offload the GPU and facilitate consistent and high frame rates.



(a) (b) (c)

Figure 11: (a) A scene showing a complex terrain dataset (consisting of approximately 1 giga voxels) generated by the method of Peytavie et al. [PGGM09] with features that cannot be represented with heightmap-based approaches. (b) A distant view on another terrain with many caves which was used for our performance tests. (c) A wireframe rendering of a terrain model generated by our system.

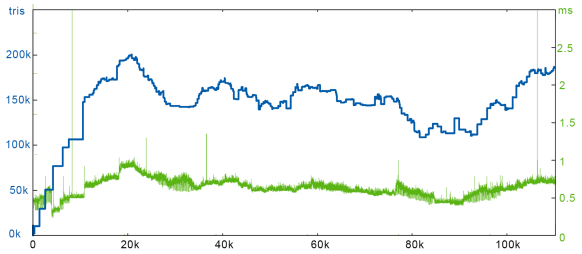


Figure 12: Performance graph for a typical fly-through of the terrain shown in Figure 11b. The graph shows the triangle count (blue) and the total frame time (green) over the frame number. The screen resolution was set to 1920×1080.

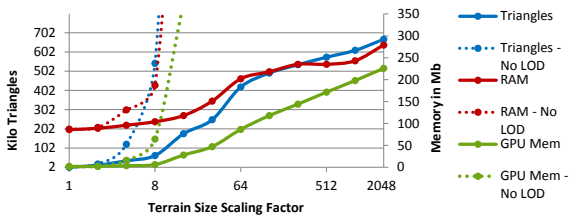


Figure 13: The plot shows how the memory consumption of the LOD system and the triangle count of the terrain mesh grow with increasing size of the terrain (solid lines). Dashed lines depict the resource consumption of the same dataset without LOD.

5.1. Volumetric Terrain

Figure 11 shows real-time renderings of a complex terrain with arches and overhangs. The coarse shape is stored in a discrete density field which is enhanced by small scale procedural geometry at run-time. This results in detailed shapes and good control over the overall appearance while maintaining a small memory footprint. The terrain surface is textured using triplanar mapping along the coordinate system axes similar to [PGGM09,LMS11]. We shade the terrain using the Phong shading model and tangent space normal mapping to further enhance detail beyond triangle level. Note that tangent and binormal vectors do not need to be stored explicitly because the triplanar mapping approach allows their reconstruction within the shader. To add additional variations and break repetitive texture patterns, we further apply a two-dimensional colormap that is addressed by terrain height and slope.

Figure 12 shows the relationship between triangle count and frame time for a typical fly-through with different speeds. The lattice size of a single hexahedral block of the volume mesh was set to $16 \times 16 \times 16$ which provides a good balance between adaptivity and performance. We achieve an average refresh rate above 1500 Hz without any culling algorithm. Simple view frustum culling could reduce the number of triangles by a estimated factor of about 3 to 6 depending on the scene and viewing conditions. Additional occlusion culling could further improve performance especially in caves and deep valleys. The high rendering speed leaves enough room to increase the geometric detail as well as for other GPU intensive tasks (e.g. more elaborate shading or procedural texturing). This makes our system well-suited for demanding real-time applications where terrain rendering is only allowed to consume a small part of the overall frame time. We attribute the high frame rates mainly to two facts: First, the good triangle batch sizes which reduces the impact of API overhead. Second, the GPU is only responsible for rendering and does not need to perform stitching operations at cell interfaces as required in previous methods. The mesh generated by our algorithm also exhibits good triangle quality as can be seen in Figure 11c.

Figure 13 demonstrates the scalability of our LOD method. For testing we increased the size of the domain cube while preserving a constant mesh resolution at the location of the observer. Since the resource consumption can vary widely across different terrain shapes, we have also shown measurements without LOD (dashed lines) for comparison.

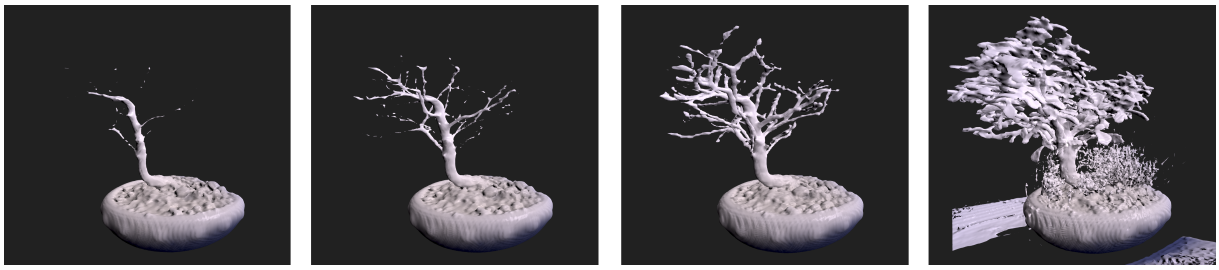


Figure 14: The Bonsai dataset rendering using our LOD algorithm with different isovalues. The isovalue has been changed during run-time, an update requires approximately 1 second. Note that the update runs in a background thread on the CPU.

Note that the x-axis has a logarithmic scale while the y-axis has a linear scale. As can be seen, our LOD algorithm achieves a memory and rendering complexity of $\mathcal{O}(\log n)$ with n being the edge length of the domain cube. The computational cost for the surface extraction heavily depends on the movement speed of the viewer but scales equally well with increasing terrain sizes. As indicated by the dashed lines, bottom-up techniques that require the generation of a full resolution mesh during preprocessing would consume massive amounts of memory and computation time, making them impractical for vast terrain models.

5.2. Indirect Volume Visualization

The interactive visualization of volume data poses different requirements to the rendering system than the display of terrain models. The viewer needs to move freely and quickly through the entire scene instead of traversing the dataset slowly near the isosurface. There are usually no other time consuming rendering tasks and frame rates of approximately 30 Hz are often sufficient. The desired distribution of geometric detail is also different to terrain visualization applications, where a distance driven LOD system yields good results. Instead the definition of one or more areas of interest (AOI) is often used to control refinement.

Our LOD algorithm is flexible enough to fulfill the above requirements and can thus be used to explore and to analyze huge volumetric datasets interactively. As the surface extraction is performed incrementally in a background thread, the rendering loop always stays responsive even when fast viewpoint changes or sudden AOI modifications induce large amounts of refinement operations. In such a scenario the refinement down to the requested detail level is distributed and carried out over several frames.

One advantage of our algorithm, compared to bottom-up methods, is that it can be combined with run-time updates of the isosurface as required by applications where the isovalue needs to be changed. To update the entire mesh at once we recompute the isosurface in each cell of the active front but keep the current refinement state of the diamond hierarchy. This feature is demonstrated in Figure 14 where a CT dataset

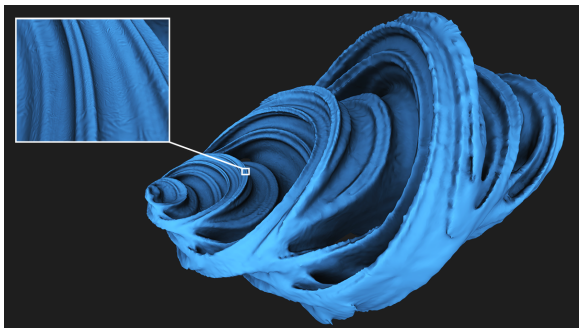


Figure 15: A Julia set fractal rendered with our approach. The entire geometry was procedurally generated during runtime. The closeup on the top-left shows the amount of detail at the refinement point.

is rendered with different isovalues. In our current implementation the updates of the whole surface cannot be performed fast enough to display animated isosurfaces interactively as a typical update takes about a second. In principle, the algorithm can be accelerated by moving the surface extraction phase to the GPU [DZTS07], where we expect it to reach interactive speeds. In that case, however, the additional workload could interfere with rendering, leading to less stable frame rates (depending on the application this might of course be acceptable). Since the surface is computed on demand, we are also able to visualize procedurally generated content without the need to construct the entire geometry at a high resolution. In Figure 15 we demonstrate this for a Julia set fractal.

The average timings for the different parts of the surface extraction step are given in Table 1. As can be seen, the sampling of the scalar function is the most expensive step. This can vary strongly depending on the volumetric models. The time required for all other steps mostly depends on the surface area inside the cell.

The effect of the triangle order optimization on rendering speed is shown in Figure 16. The overall performance gain is moderate and might be dominated by rasterization

Algorithm Stage	Abs. Time	Rel. Time
scalar function sampling	1.510 ms	50.2%
marching cubes	0.102 ms	3.3%
mesh simplification	0.868 ms	28.9%
mesh smoothing	0.015 ms	0.4%
normal computation	0.507 ms	16.8%
total	3.002 ms	-

Table 1: Average timings for different steps of the surface extraction for a single cell containing four lattices with $16 \times 16 \times 16$ hexahedral elements each.

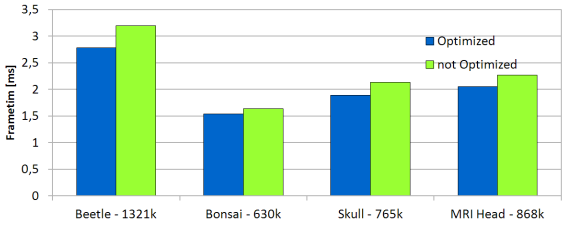


Figure 16: Performance improvement of the triangle order optimization for various datasets.

and shading costs in most applications. However, the order optimization of hexahedral elements inside the volumetric mesh comes without any additional run-time overhead and is easy to implement.

Our algorithm can be seen as a combination of purely tetrahedra-based surface extraction algorithms and the marching cubes algorithm. It is therefore interesting how the proposed method compares to these other techniques in terms of surface approximation capabilities. For each algorithm we measured the geometric error for different mesh resolutions by uniformly sampling the surface with a fixed number of samples and computing the RMS of Hausdorff distances from each sample to a high resolution ground-truth mesh. The results for the stag beetle dataset are shown in Figure 17. In this test our method outperforms tetrahedra-based approaches and is comparable to MC despite the cell distortion inherent to our subdivision scheme.

Discussion Although terrain visualization was the initial focus of our view-dependent level of detail algorithm, it is able to render large arbitrary isosurfaces at high frame rates. It is computationally efficient and can handle procedurally generated content as well as isovalue changes at run-time. Direct volume rendering methods can also render (multiple) isosurfaces, but also support the rendering of semi-transparent regions. However, the evaluation (marching along rays, transfer functions, etc.) can be very expensive and acceleration data structures become necessary. As these data structures are usually built prior to rendering, dynamic and online generated datasets are typically hard to visualize with such methods. The high evaluation cost of ray

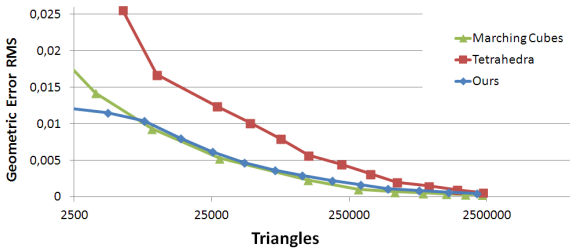


Figure 17: The geometric error (stag beetle dataset) as a function of triangle count for different surface extraction methods. For comparison, a uniform detail level was chosen. Red: the tetrahedra-based method by Gregorski et al. [GDL*02]. Green: marching cubes algorithm. Blue: our method.

casting becomes even more apparent as display resolution increases. As our method is based on rasterization it allows to trade speed for quality more easily and can render complex isosurfaces at low cost. Visibility queries which are required for shadowing techniques can also be efficiently realized through rasterization, whereas direct volume rendering methods have to resort to expensive secondary rays. However, ray marching enables global illumination effects more easily (e.g. [JSYR12, WKSD13]). Note that rasterization methods also allow the decoupling of the shading frequency from the pixel resolution, e.g. by performing shading computation per vertex, or using more elaborate approaches [LD12]. Rendering of triangle meshes can also benefit from hardware anti-aliasing, while ray casting-based methods might require prefiltering techniques to avoid aliasing without excessive supersampling [YLM06].

Our approach also has some limitations. Due to the lack of a globally unique surface parametrization, only procedural or object-space (e.g. [LD07]) texturing methods can be applied. In particular for terrain rendering applications, e.g. games or simulators, often carefully authored, unique global textures are preferred for best quality and realism. Such (out-of-core) texturing algorithms would require adaptation to our terrain rendering method. We also cannot encode geometric information of higher detail levels in normal maps because we have no information about the correspondence of surfaces from different detail levels.

Lastly, the transitions in our LOD algorithm might be visible. Similar to previous works we do not provide geomorphing due to possible topological changes between detail levels. Using alpha blending for the transition between detail levels constitutes one possible solution [GW07].

Since we generate the geometry on-the-fly, the scalar function has to be sampled at run-time. This may lead to delayed LOD updates if the scalar function is very complex and its evaluation is time consuming. Nevertheless, the rendered mesh is always consistent but the LOD hierarchy might not

update rapidly enough to accommodate fast viewer movements. Evaluating the scalar function in parallel can mitigate this problem. Lastly, as our approach is independent from the actual implementation of the surface extraction algorithm, we can easily combine it with other isosurface triangulation methods operating on regular cubical grids. For example, it can be combined with fast GPU-based marching cubes [DZTS07, SDC09] to accelerate LOD updates.

6. Conclusion and Future Work

We presented a view-dependent level of detail algorithm for real-time rendering of large, detailed isosurfaces and demonstrated its application to scientific datasets, volumetric terrain, and procedural generated isosurfaces. Unlike previous methods like massive model visualization techniques, our algorithm does not require any preprocessing. At the same time it can be used in performance critical scenarios where ray casting algorithms are still to slow. Our method also allows to update even large isosurfaces during run-time within a few seconds, which is useful for interactive inspection of volumetric datasets. Moreover, we introduced a sampling scheme that reduces aliasing artifacts and produces consistent results across cell borders. Finally, it was shown that our approach handles nonlinear scalar functions better than tetrahedra-based methods and performs equally well as the marching cubes algorithm which does not support view-dependent level of detail.

Our algorithm can be improved by porting the surface extraction stage to the GPU. This is expected to allow updates of the entire isosurface at interactive rates which is especially interesting for data visualization applications that need to render animated time varying volumetric datasets.

Acknowledgments The work of Manuel Scholz and Jan Bender was supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School CE at TU Darmstadt. Special thanks go to Peytavie Adrien and Eric Galin for providing us complex terrain datasets.

References

[Áfr12] ÁFRA A. T.: Interactive ray tracing of large models using voxel hierarchies. *Computer Graphics Forum 31*, 1 (2012), 75–88. 3

[AGD10] AMMANN L., GÉNEVAUX O., DISCHLER J.-M.: Hybrid rendering of dynamic heightfields using ray-casting and mesh rasterization. In *Proc. Graphics Interface* (2010), pp. 161–168. 3

[BGB*05] BORGEAT L., GODIN G., BLAIS F., MASSICOTTE P., LAHANIER C.: GoLD: interactive display of huge colored and textured models. *ACM Transactions on Graphics 24*, 3 (2005), 869–877. 3

[BGP09] BOESCH J., GOSWAMI P., PAJAROLA R.: RASTeR: Simple and efficient terrain rendering on the GPU. In *Proc. Eurographics - Areas Papers* (2009), pp. 35–42. 3

[CGG*03] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: BDAM: Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (2003), 505–514. 3

[CGG*04] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics 23*, 3 (2004), 796–803. 3

[CMPS96] CIGNONI P., MONTANI C., PUPPO E., SCOPIGNO R.: Optimal isosurface extraction from irregular volume data. In *Proc. Symposium on Volume Visualization* (1996), pp. 31–38. 2

[CMS06] CARR H., MOLLER T., SNOEYINK J.: Artifacts caused by simplicial subdivision. *IEEE Transaction on Visualization and Computer Graphics 12*, 2 (2006), 231–242. 5

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2009). 3

[Cra12] CRASSIN C.: Dynamic sparse voxel octrees for next-gen real-time rendering, 2012. ACM SIGGRAPH Course "Beyond Programmable Shading". 4

[Dac06] DACHSBACHER C.: *Interactive Terrain Rendering – Towards Realism with Procedural Models and Graphics Hardware*. PhD thesis, University of Erlangen-Nuremberg, 2006. 3

[DKW09] DICK C., KRÜGER J., WESTERMANN R.: GPU ray-casting for scalable terrain rendering. In *Proc. Eurographics - Areas Papers* (2009), pp. 43–50. 3

[DKW10] DICK C., KRUEGER J., WESTERMANN R.: GPU-aware hybrid terrain rendering. In *Proc. IADIS Computer Graphics, Visualization, Computer Vision and Image Processing* (2010), pp. 3–10. 3

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: ROAMing terrain: real-time optimally adapting meshes. In *Proc. IEEE Visualization* (1997), pp. 81–88. 3

[DZTS07] DYKEN C., ZIEGLER G., THEOBALT C., SEIDEL H.-P.: *GPU marching cubes on shader model 3.0 and 4.0*. Research Report MPI-I-2007-4-006, Max-Planck-Institut für Informatik, August 2007. 10, 12

[GDL*02] GREGORSKI B., DUCHAINEAU M., LINDSTROM P., PASCUCCI V., JOY K. I.: Interactive view-dependent rendering of large isosurfaces. In *Proc. IEEE Visualization* (2002), pp. 475–484. 3, 5, 11

[Ger02] GERSTNER T.: Multiresolution extraction and rendering of transparent isosurfaces. *Computers & Graphics 26*, 2 (2002), 219–228. 2

[GM05] GOBBETTI E., MARTON F.: Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics 24*, 3 (2005), 878–885. 3, 4

[GMC*06] GOBBETTI E., MARTON F., CIGNONI P., DI BENEDETTO M., GANOVELLI F.: C-BDAM - compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum 25*, 3 (2006). 3

[GMG08] GOBBETTI E., MARTON F., GUITIÁN J. A. I.: A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *Visual Computer 24*, 7 (2008), 797–806. 3

[GR99] GERSTNER T., RUMPF M.: Multiresolutional parallel

isosurface extraction based on tetrahedral bisection. In *Proc. Symposium Volume Visualization* (1999), pp. 267–278. 2

[GW07] GIEGL M., WIMMER M.: Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. *Computer Graphics Forum 26*, 1 (2007), 46–49. 8, 11

[Hop96] HOPPE H.: Progressive Meshes. *Proc. SIGGRAPH* (1996), 99–108. 4

[HSH09] HU L., SANDER P. V., HOPPE H.: Parallel view-dependent refinement of progressive meshes. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2009), pp. 169–176. 4

[JL12] JUDNICH J., LING N.: Symmetric cluster set level of detail for real-time terrain rendering. In *Multimedia and Expo (ICME)* (2012), pp. 320–324. 3

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of Hermite data. *ACM Transactions on Graphics 21*, 3 (2002), 339–346. 2

[JSYR12] JÖNSSON D., SUNDÉN E., YNNERMAN A., ROPINSKI T.: Interactive volume rendering with volumetric illumination. In *Eurographics STAR program* (2012). 11

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. SIGGRAPH* (1987), pp. 163–169. 1, 2

[LD07] LEFEBVRE S., DACHSBACHER C.: Tiletrees. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2007), pp. 25–31. 11

[LD12] LIKTOR G., DACHSBACHER C.: Decoupled deferred shading for hardware rasterization. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2012), pp. 143–150. 11

[Len10] LENGYEL E. S.: *Voxel-based Terrain for Real-time Virtual Simulations*. PhD thesis, University of California at Davis, Davis, CA, USA, 2010. 4

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics 23*, 3 (2004), 769–776. 3

[LKES09] LIVNY Y., KOGAN Z., EL-SANA J.: Seamless patches for GPU-based terrain rendering. *The Visual Computer 25*, 3 (2009), 197–208. 3

[LKR*96] LINDSTROM P., KOLLER D., RIBARSKY W., HODGES L. F., FAUST N., TURNER G. A.: Real-time, continuous level of detail rendering of height fields. In *Proc. SIGGRAPH* (1996), pp. 109–118. 3

[LMS11] LOEFFLER F., MUELLER A., SCHUMANN H.: Real-time rendering of stack-based terrains. In *Proc. Vision, Modelling and Visualization* (2011), pp. 161–168. 4, 9

[LPT03] LARIO R., PAJAROLA R., TIRADO F.: Hyperblock-quadtin: Hyper-block quadtree based triangulated irregular networks. In *Proc. IASTED Visualization, Imaging and Image Processing* (2003), pp. 733–738. 3

[NY06] NEWMAN T. S., YI H.: A survey of the marching cubes algorithm. *Computers & Graphics* (2006), 854–879. 3

[Pas04] PASCUCCI V.: Isosurface computation made simple: hardware acceleration, adaptive refinement and tetrahedral stripping. In *Proc. Joint Eurographics-IEEE TVCG Symposium on Visualization* (2004), pp. 293–300. 3

[PG07] PAJAROLA R., GOBBETTI E.: Survey of semi-regular multiresolution models for interactive terrain rendering. *Visual Computer 23*, 8 (2007), 583–605. 3

[PGGM09] PEYTAVIE A., GALIN E., GROSJEAN J., MÉRILLOU S.: Arches: a framework for modeling complex terrains. *Computer Graphics Forum 28*, 2 (2009), 457–467. 1, 2, 4, 9

[PT90] PAYNE B. A., TOGA A. W.: Surface mapping brain function on 3d models. *IEEE Computer Graphics & Applications 10*, 5 (1990), 33–41. 2

[RCBW12] REICHL F., CHAJDAS M. G., BÜRGER K., WESTERMANN R.: Hybrid sample-based surface rendering. In *Proc. Vision, Modelling and Visualization* (2012), pp. 47–54. 3

[RMD11] REINER T., MÜCKL G., DACHSBACHER C.: Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computers & Graphics 35* (2011), 596–603. 1

[SDC09] SCHMITZ L. A., DIETRICH C. A., COMBA J. L. D.: Efficient and high quality contouring of isosurfaces on uniform grids. In *Computer Graphics and Image Processing (SIBGRAPI)* (2009), pp. 64–71. 12

[SM05] SANDER P. V., MITCHELL J. L.: Progressive buffers: View-dependent geometry and texture for LOD rendering. In *Proc. Symposium on Geometry Processing* (2005), pp. 129–138. 3

[ST90] SHIRLEY P., TUCHMAN A.: A polygonal approximation to direct scalar volume rendering. In *Computer Graphics* (1990), pp. 63–70. 2

[Str09] STRUGAR F.: Continuous distance-dependent level of detail for rendering heightmaps. *Journal of Graphics, GPU, and Game Tools 14*, 4 (2009), 57–74. 3

[Vir14] Virtual Terrain Project, 2014. URL: http://www.vterrain.org. 3

[WCM12] WEBER G. H., CHILDS H., MEREDITH J. S.: Efficient parallel extraction of crack-free isosurfaces from adaptive mesh refinement (AMR) data. In *Proc. Symposium on Large Data Analysis and Visualization* (2012), pp. 31–38. 2

[WDF08] WEISS K., DE FLORIANI L.: Multiresolution interval volume meshes. In *Proc. Symposium on Volume and Point-Based Graphics* (2008), pp. 65–72. 4, 5

[WF11] WEISS K., FLORIANI L. D.: Simplex and diamond hierarchies: models and applications. *Computer Graphics Forum 30*, 8 (2011), 2127–2155. 5

[WK02] WU J., KOBBELT L.: Fast mesh decimation by multiple-choice techniques. In *Proc. Vision, Modelling and Visualization* (2002), pp. 241–248. 7

[WKE99] WESTERMANN R., KOBBELT L., ERTL T.: Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer 15* (1999), 100–111. 2

[WKL*01] WEBER G. H., KREYLOS O., LIGOCKI T. J., SHALF J. M., HAGEN H., HAMANN B.: Extraction of crack-free isosurfaces from adaptive mesh refinement data. In *Data Visualization* (2001), Springer Verlag, pp. 25–34. 2

[WKSD13] WEBER C., KAPLANYAN A., STAMMINGER M., DACHSBACHER C.: Interactive direct volume rendering with many-light methods and transmittance caching. In *Proc. of Vision, Modeling and Visualization* (2013), pp. 195–202. 11

[YLM06] YOON S.-E., LAUTERBACH C., MANOCHA D.: R-LODs: fast LOD-based ray tracing of massive models. *The Visual Computer 22*, 9 (2006), 772–784. 3, 11

[ZQHK04] ZHANG N., QU H., HONG W., KAUFMAN A. E.: Shic: A view-dependent rendering framework for isosurfaces. In *Volume Visualization and Graphics* (2004), IEEE Computer Society, pp. 63–70. 3