Vision, Modeling, and Visualization (2013)
Michael Bronstein, Jean Favre, and Kai Hormann (Eds.)

Level of Detail for Real-Time Volumetric Terrain Rendering

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

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

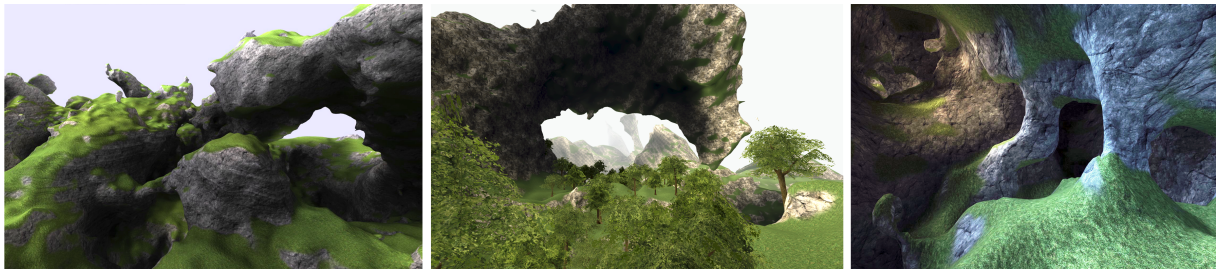


Figure 1: Screenshots from our real-time terrain rendering system. The above scenes show complex features like caves, arches, and steep cliffs, and were rendered with a resolution of 1920x1080 at approximately 1500 Hz.

Abstract

Terrain rendering is an important component of many GIS applications and simulators. Most methods rely on heightmap-based terrain which is simple to acquire and handle, but has limited capabilities for modeling features like caves, steep cliffs, or overhangs. In contrast, volumetric terrain models, e.g. based on isosurfaces can represent arbitrary topology. In this paper, we present a fast, practical and GPU-friendly level of detail algorithm for large scale volumetric terrain that is specifically designed for real-time rendering applications. Our algorithm is based on a longest edge bisection (LEB) scheme. The resulting tetrahedral cells are subdivided into four hexahedra, which form the domain for a subsequent isosurface extraction step. The algorithm can be used with arbitrary volumetric models such as signed distance fields, which can be generated from triangle meshes or discrete volume data sets. In contrast to previous methods our algorithm does not require any stitching between detail levels. It generates crack free surfaces with a good triangle quality. Furthermore, we efficiently extract the geometry at runtime and require no preprocessing, which allows us to render infinite procedural content with low memory consumption.

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

Since the beginning of computer graphics an important goal is to model, simulate and render large natural scenes. A key element of outdoor scenes is the terrain itself. Efficient terrain rendering has therefore been in focus of research for many years, and has a wide range of applications in virtual reality, simulators, geographic information system and games. To render large scale terrain models interactively, a level of detail (LOD) system is required to invest memory and computational resources only where necessary.

Most research has focused on heightmap-based approaches which are well established and provide a good tool to render 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 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.

Recently, volumetric approaches have gained more attention. They can represent arbitrary shapes and thus provide more flexibility than heightmaps. The terrain surface is defined by a density function from which an isosurface is extracted and rendered. The downside is the high memory consumption of volumetric data sets and that more elaborate level of detail algorithms than for heightmap-based terrain are required. Previous work in this field either focused on (rather complicated) stitching algorithms for detail level transitions, or on modeling tools and data representations. We focus on efficient rendering and present a novel level of detail algorithm for volumetric terrain that requires no stitching and can extract the terrain surface from arbitrary volumetric models at runtime on the CPU. Unlike massive model visualization techniques our approach does not require any preprocessing and allows us to render infinite procedural content with low memory usage. Our approach uses hexahedral cells to extract the surface. 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. Finally, we also discuss aliasing problems in the context of level of detail rendering and provide a sampling strategy that is applicable to many volumetric models. Three examples of complex procedural terrain, rendered with our method, are shown in Figure 1.

2. Related Work

Methods for rendering large scale terrain data can be roughly categorized into heightmap-based methods, (generic) massive model visualization techniques, and volumetric terrain rendering. We briefly discuss the pros and cons of existing approaches with respect to the goals of this work.

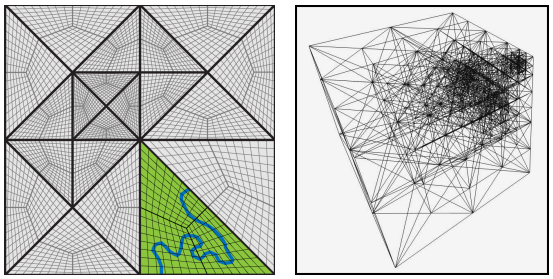
Heightmap-Based Terrain Rendering has almost been studied since the beginning of computer graphics. Previous work spans the entire range from fine grained level of detail methods that operate on the individual triangles, e.g. [LKR*96, DWS*97], to coarse grained methods which became the first choice with the advent of powerful graphics hardware. The latter adjust the detail level on the granularity of large chunks of geometry and adapt their borders to match the resolution of neighbor chunks, e.g. [LH04, Str09, BGP09, LKES09]. For further detail we refer to surveys on (multiresolution) terrain models [Dac06, PG07]. 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 are designed to render arbitrary highly detailed models and thus can be used for rendering terrains as well. 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. In [CGG*04, SM05, BGB*05] the geometry is stored in a tree data structure where each node contains a part of the geometry at a certain level of detail. 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. Gobbetti and Marton [GM05] extend this approach by switching to a precomputed voxel representation of the geometry if the screen space footprint of a node becomes smaller than few pixels. In [CGG*03, LPT03] methods for rendering terrain with triangulated irregular networks (TINs) are presented. Although only demonstrated for heightmap datasets only, TINs can also represent overhangs and other complex terrain features. Hu et al. [HSH09] extend the idea of progressive meshes [Hop96] with a view dependent refinement scheme. However, it requires a significant amount of GPU resources for the hierarchy management. Lastly there are also ray-tracing algorithms, e.g. [YLM06, GMG08, Áfr12] that run at interactive frame rates but they are still too slow for highly interactive real-time applications.

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 has not been studied as extensively as the previous classes, and only gained more attention in the last few years. 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 completely independent and no stitching is required, which avoids all these problems. Note that stack-based terrain representations can also be used with our LOD algorithm.

There are also solutions that focus on level of detail rendering of arbitrary isosurfaces, e.g. [GDL*02, Pas04]. They utilize an adaptive longest edge bisection hierarchy to build a conforming tetrahedral mesh on which the isosurface is extracted. Compared to the method described in this work, extracting a surface directly from these adaptive tetrahedral meshes results in significantly more triangles and a lower mesh quality.



(a) (b)

Figure 2: (a) The concept of our LOD algorithm in 2D: Thick black lines show the 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. (b) A longest edge bisection tetrahedra hierarchy in 3D.

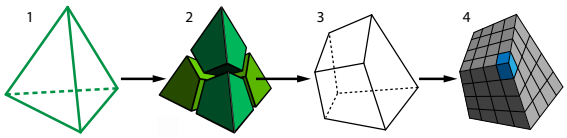


Figure 3: A tetrahedron (1) in the LEB hierarchy is split into four hexahedra (2). Each of the hexahedra (3) is regularly subdivided into a lattice (4) of hexahedral elements (blue). These are used to extract the surface using the Marching Cubes algorithm.

3. Level Of Detail Algorithm

Our approach is based on density functions to represent (large scale) volumetric terrain. The terrain surface is defined as an isosurface which is approximated by a triangular mesh and which can be efficiently rendered on graphics hardware.

An essential component of real-time terrain rendering systems is the level of detail algorithm which is usually based on a hierarchical decomposition of the domain space. To our knowledge, all previous volumetric terrain rendering methods use an octree to subdivide space into cubical cells of different sizes. 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. Our method avoids this problem by using a conforming hexahedral mesh instead of an octree decomposition. By definition, faces of neighboring cells in a conforming mesh always coincide. Therefore, the lattices used during surface extraction match at cell boundaries and the extracted surface exhibits no cracks in its triangulation (see Figure 2).

To allow interactive traversal of the terrain, the hexahedral mesh has to adapt quickly to new viewer positions. We fulfill this requirement in two steps: First, a diamond hierarchy [WDF08] is used to efficiently build an adaptive conforming tetrahedral partitioning 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 creates better isosurface approximations with less artifacts compared to simplex based methods [CMS06] and the resulting meshes have a lower number of triangles and a better mesh topology with a more evenly distributed vertex valence. Note that these advantages remain even though we use deformed hexahedral domains instead of cubic domains.

For the surface extraction step we propose a novel sampling strategy which maintains the independence of cells and avoids aliasing. It is based on a frequency space decomposition of the density 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 explains the construction of the density function and its sampling. Figure 4 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 terrain mesh to the current viewing conditions. Our LOD algorithm uses three dimensional diamond hierarchies [WF11] which are based on the longest edge bisection 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 emerge. This is crucial for obtaining terrain 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 simulta-

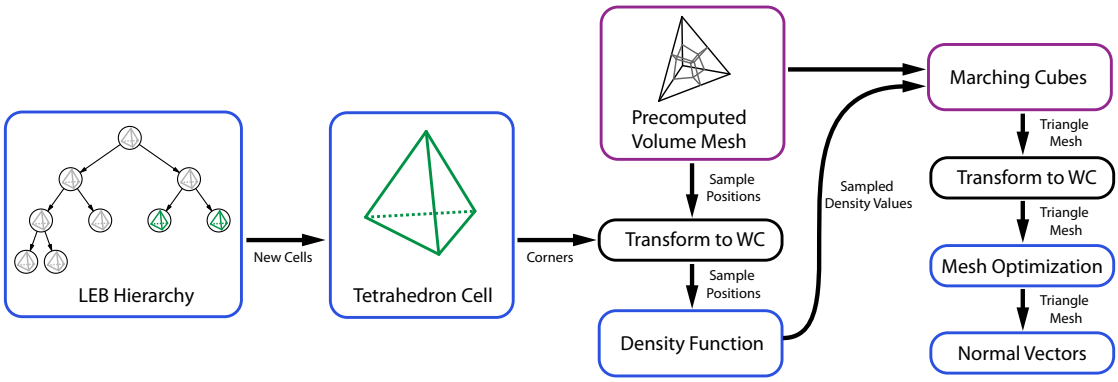


Figure 4: Algorithm overview: Blue frames denote steps that operate in Cartesian world coordinates (WC), while steps with purple frames operate in barycentric coordinates. When the hierarchy is refined, new cells are passed to the surface extraction step. Its corner positions are used to transform sample and vertex positions from barycentric coordinates into world space.

neously to maintain a conforming tetrahedralization. We use the encoding scheme by Weiss and De Floriani [WDF08] 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.

Our prototype implementation uses a simple distance-based refinement criterion. For each node the distance d of its bounding sphere to the viewer is computed. The target refinement level l_t is then defined as follows:

$$l_t = \log_a(d). \tag{1}$$

A node is split whenever its target level l_t is larger than its level l_h inside the longest edge bisection hierarchy. The parameter a controls the relationship between detail level and distance and should be chosen according to the field of view of the viewer. We also add a small hysteresis of at least one hierarchy level to control the collapsing of nodes.

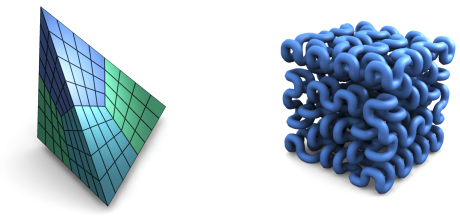
3.2. Surface Extraction

When a cell in the level of detail hierarchy is created, the surface inside this new cell has to be extracted from the density 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 surface inside 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 runtime. 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, a volume mesh has the shape of a tetrahedra (which we generate during our level of detail algorithm) containing hexahedral lattices (see Figure 5a). Since all cells of the LOD hierarchy are subdivided in the same way, we can reuse a single volume mesh which can be precomputed at startup. To precompute this volume mesh we first split a tetrahedra into four hexahedra H_i. Each of them can be further subdivided into a regular lattice of smaller (4^3 in our case) hexahedral elements. The elements of all four hexahedra H_i are then unified into a single volume mesh representation (see Figure 5a), 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 5b).

Later, the Marching Cubes algorithm uses the same element order to extract the isosurface. This results in a spatially coherent triangle mesh layout which improves vertex cache utilization during rendering. Note that we store the the volume mesh in four dimensional barycentric coordinates, which allows to reuse it for all tetrahedral cells independent from the actual shape, position and rotation.



(a) (b)

Figure 5: (a) The volume mesh contains all four lattices with $4 \times 4 \times 4$ hexahedral elements each. (b) We use a three dimensional Hilbert curve to reorder hexahedral elements for improved locality inside the hexahedra of the volume mesh.

The modified MC algorithm can be described in three steps: First, the density function is evaluated at each vertex position of the volume mesh. Second, each edge of the volume mesh is inspected. If a sign change 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. Additionally, 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 density 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.

3.3. Density Function

We use the density function to define the shape of the terrain. Our LOD algorithm does not impose any restrictions on this function. Nevertheless, the discrete sampling of the density function during the surface extraction phase can lead to aliasing which degrades visual quality and make LOD transitions noticeable. To mitigate this problem we apply a spatially-varying low-pass filter to the density function during sampling. A naive implementation as a discrete filter is very general, but would require prohibitively many evaluations of the density function.

To accelerate filtering, we express our density function 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 density 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 $\mathbf{x}$ 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 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 LOD levels that are not noticeable in practice.

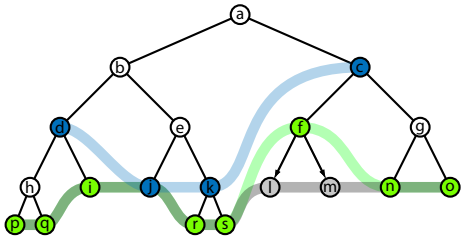


Figure 6: Visualization of the active front (green) and the GPU-buffer front (blue). The gray line depicts the active front after node f is split.

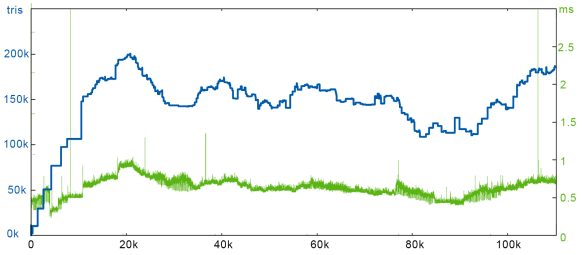


Figure 7: Performance graph for a typical fly-through of the terrain shown in Figure 8b. The graph shows the triangle count (blue) and the total frame time (green) over the frame number. The screen resolution was set to 1920x1080.

4. Rendering

In this section we explain how to efficiently render the geometry generated in the surface extraction phase. When the viewer explores the terrain, the active front nodes are constantly created and destroyed (cf. Section 3.1). The number of triangles and vertices generated for new nodes may vary strongly depending on the surface area of the isosurface inside the cells. This is a problem for rendering as GPU resource allocation during runtime is typically slow and drawing many small triangle meshes is inefficient on modern graphics hardware.

To overcome these problems we introduce an additional front into the LEB hierarchy denoted as the *GPU buffer front* (see Figure 6). A node 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 a GPU-buffer front node, it is split into two child nodes of equal size and the front moves downwards. When the geometry of two sibling nodes fits into a single GPU-buffer front node, the two nodes 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 API draw call count is decreased. Second, the vertex and index buffers are all of the same size and can therefore be reused easily; this avoids costly GPU resource allocation at runtime.

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

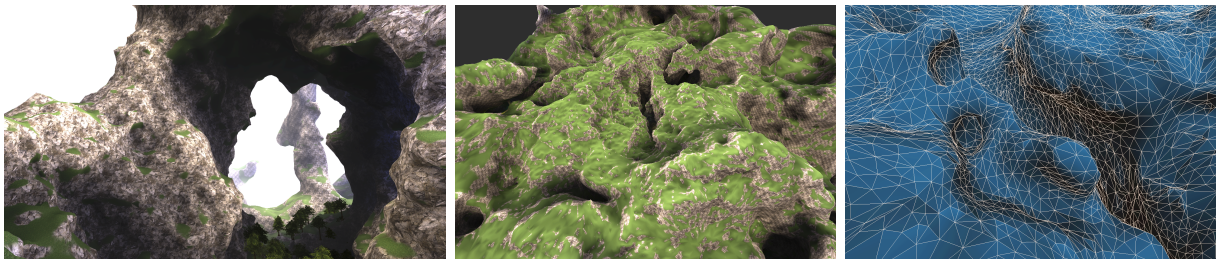
In our example, the terrain is textured using triplanar mapping along the coordinate system axis similar to [LMS11, PGGM09]. We also use a two dimensional colormap that is addressed by height and slope which introduces more variations and breaks repetitive texturing patterns. To make the LOD transitions less visible, many heightmap-based terrain systems use geomorphing. This is not possible with volume-based terrain representations, as topological changes of the surface may arise between different LOD levels. Image space blending as discussed in [GW07] provides a viable alternative that is commonly used in industry and has shown to work well in practice.

5. Results

We implemented our algorithm in C++ and used the DirectX 11 API for rendering. All tests where carried out on a quad core AMD Phenom II 3.2 GHz processor and an NVIDIA GeForce GTX 580 graphics card. Remember that surface extraction and hierarchy updates are performed in a background thread on the CPU to offload the GPU and facilitate consistent and high frame rates.

Figure 7 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 16x16x16 which provides a good balance between adaptivity and performance. We achieve an average refresh rate above 1500 Hz without any culling. Simple view frustum culling could reduce the number of triangles by a 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 and for other GPU intensive tasks. 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 the spatial coherent layout of the volume mesh which leads to vertex cache friendly triangulations and to good triangle batch sizes. The mesh generated by our algorithm also exhibits good triangle quality as can be seen in Figure 8c.

Figure 9 demonstrates the scalability of our LOD method. For the test 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. Note that



(a) (b) (c)

Figure 8: (a) A scene showing complex terrain with features that can not be represented with heightmap-based approaches. (b) Distant view onto a terrain with many caves which was used for our performance tests. (c) Wireframe rendering of a terrain model generated by our system.

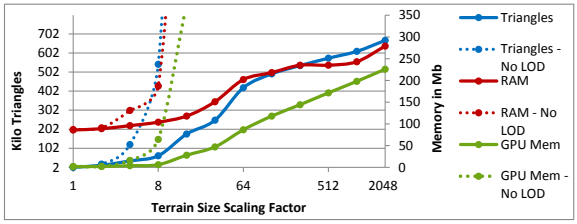


Figure 9: The plot shows how the memory consumption of the LOD system and the triangle count of the terrain mesh grow with increasing size of the terrain (solid lines). Dashed lines depict the resource consumption of the same dataset without LOD.

Algorithm Stage	Abs. Time	Rel. Time
density 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 parts of the surface extraction step for a single node (four $16 \times 16 \times 16$ blocks).

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 depends heavily 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.

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

Discussion Our level of detail algorithm can be used to efficiently render complex terrain with geometric features that are impossible to represent with heightmap-based methods (see Figure 8(a)). Nevertheless, our approach has some limitations. The presented algorithm does not create a parameterization of the terrain surface that is consistent across detail levels. Therefore, a unique mapping of high resolution textures to the terrain surface, as it is common for heightmap-based terrain, is not possible. We also can not encode geometric information of higher detail levels in normal maps because we have no information about the correspondence of surfaces from different detail levels. Note that we share these limitations with previous approaches and efficient texturing of volumetric terrain is still an open problem.

Another disadvantage are popping artifacts which make LOD transitions more visible. Similar to previous works we did not find a way to efficiently implement geomorphing due to possible topological changes between detail levels. Using alpha blending for the transition between detail levels provides a possible solution but comes at the cost of lower rendering performance.

Since we generate the geometry on the fly, the density function has to be sampled during runtime. This may lead to delayed LOD updates if the density 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 density function in parallel can mitigate this problem.

6. Conclusion and Future Work

We presented a fast and GPU friendly level of detail algorithm for real-time rendering of volumetric terrain which can be used with arbitrary volumetric models. In contrast to previous methods our algorithm does not require any preprocessing and can handle infinite procedural terrain of arbitrary detail. Additionally, a sampling scheme was introduced that avoids aliasing artifacts and produces consistent results across cell borders.

We applied a simple texturing method to the terrain which is commonly used for isosurfaces. However, we believe that a more sophisticated approach will benefit visual quality. To create more natural looking terrain models, we would also like to do an in-depth investigation of different types of density functions and procedural modeling techniques.

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.

References

[Áfr12] ÁFRA A. T.: Interactive ray tracing of large models using voxel hierarchies. *Computer Graphics Forum 31*, 1 (2012), 75–88. 2

[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. 2

[BGP09] BOESCH J., GOSWAMI P., PAJAROLA R.: RASTeR: Simple and efficient terrain rendering on the GPU. In *Proc. EUROGRAPHICS Areas Papers, Scientific Visulization* (2009), pp. 35–42. 2

[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 (sept 2003), 505–514. 2

[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 (Proc. SIGGRAPH) 23*, 3 (2004), 796–803. 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. 3

[Dac06] DACHSBACHER C.: *Interactive Terrain Rendering – Towards Realism with Procedural Models and Graphics Hardware*. PhD thesis, University of Erlangen-Nuremberg, 2006. 2

[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 *Proceedings of IEEE Visualization* (1997), pp. 81–88. 2

[GDL*02] GREGORSKI B., DUCHAINEAU M., LINDSTROM P., PASCUCCI V., JOY K. I.: Interactive view-dependent rendering of large isosurfaces. In *Proceedings of IEEE Visualization* (2002), pp. 475–484. 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 (Proc. SIGGRAPH) 24*, 3 (2005), 878–885. 2

[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. 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. 6

[Hop96] HOPPE H.: Progressive Meshes. *Proc. SIGGRAPH* (1996), 99–108. 2

[HSH09] HU L., SANDER P. V., HOPPE H.: Parallel view-dependent refinement of progressive meshes. In *Proc. Interactive 3D Graphics and Games* (2009), pp. 169–176. 2

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics (Proc. SIGGRAPH) 23*, 3 (2004), 769–776. 2

[LKES09] LIVNY Y., KOGAN Z., EL-SANA J.: Seamless patches for GPU-based terrain rendering. *Visual Computer 25*, 3 (2009), 197–208. 2

[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. 2

[LMS11] LOEFFLER F., MUELLER A., SCHUMANN H.: Real-time rendering of stack-based terrains. In *Proc. Vision, Modelling and Visualization* (2011), pp. 161–168. 2, 6

[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. 2

[Pas04] PASCUCCI V.: Isosurface computation made simple: hardware acceleration, adaptive refinement and tetrahedral stripping. In *Proc. Joint Eurographics-IEEE TVCG Symposium on Visualization (VisSym)* (2004), pp. 293–300. 2

[PG07] PAJAROLA R., GOBBETTI E.: Survey of semi-regular multiresolution models for interactive terrain rendering. *Visual Computer 23*, 8 (2007), 583–605. 2

[PGGM09] PEYTAVIE A., GALIN E., GROSJEAN J., MÉRILLOU S.: Arches: a framework for modeling complex terrains. *Computer Graphics Forum 28*, 2 (2009), 457–467. 2, 6

[SM05] SANDER P. V., MITCHELL J. L.: Progressive buffers: View-dependent geometry and texture for LOD rendering. In *Eurographics Symposium on Geometry Processing* (2005), pp. 129–138. 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. 2

[WDF08] WEISS K., DE FLORIANI L.: Multiresolution interval volume meshes. In *IEEE/EG Symposium on Volume and Point-Based Graphics* (2008), pp. 65–72. 3, 4

[WF11] WEISS K., FLORIANI L. D.: Simplex and diamond hierarchies: Models and applications. *Computer Graphics Forum 30*, 8 (2011), 2127–2155. 3

[WK02] WU J., KOBBELT L.: Fast mesh decimation by multiple-choice techniques. In *Proc. Vision, Modelling and Visualization* (2002), pp. 241–248. 5

[YLM06] YOON S.-E., LAUTERBACH C., MANOCHA D.: R-LODs: fast LOD-based ray tracing of massive models. *Visual Computer 22*, 9 (2006), 772–784. 2