Vision, Modeling, and Visualization (2011)
Peter Eisert, Konrad Polthier, and Joachim Hornegger (Eds.)

Real-time Rendering of Stack-based Terrains

F. Löffler[1] and Andreas Müller[2] and H. Schumann[1]

[1]University of Rostock, Germany
[2]Fraunhofer IGD Rostock, Germany

Abstract
Usually, terrain rendering relies on a 2D regular grid of height values, the so called height field. Height fields describe 2.5D surfaces and are not able to present complex 3D terrain features. In contrast, a 3D data representation quickly exceeds the available memory resources. To overcome this problem we apply material stacks. Material stacks combine the simplicity of 2D height fields and the extended modeling capabilities of 3D volumetric data. However, this approach requires expensive rendering and is difficult to realize in real-time.
In this paper we present an innovative real-time rendering approach of terrains relying on material stacks. Our approach is based on two major steps: First, a LoD hierarchy for material-stacks is generated. Second, during rendering a multi-staged quadrangulation pipeline extracts terrain surface from the material stacks. As a result, we achieve real-time frame rates at high resolutions.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation—Display Algorithms

1. Introduction

In the field of computer graphics, real-time rendering of terrain has been addressed widely. In general, both modeling as well as rendering rely on height fields: a 2D regular grid of height samples. Height fields are very compact and consume a relatively low amount of memory to represent large terrains. However, height fields can model 2.5D surfaces merely and thus, are innately not able to present complex terrain features like arches, overhangs, and caves (cf. Figure 1). By contrast, 3D data representations (i.e. volumes) allow for modeling features of this kind. Nonetheless, memory consumption - even for small terrains - is vastly and the rendering is comparatively expensive.
One way to deal with the memory problem is to apply *material stacks* [BF01, PGMG09]. A material stack consists of multiple layers of different materials, whereas air is handled as special material. Stacks are organized on a 2D regular grid; similar to height fields. Due to the discretization and the "run-length compression" of material (layers) this representation is very compact (see Figure 2).
In contrast to height fields, there exist no accurately measured data sets and merely very few procedural modeling tools. Hence, for obtaining customized 3D terrain data, interactive 3D modeling is necessary. Moreover, both interactive walkthroughs and exploration of large terrains are key objectives of entertainment applications. Consequently, a real-time rendering approach for stack-based terrain is necessary. Nonetheless, rendering of material stacks is very complex, since the implicit surface needs to be extracted. There exist very efficient surface extraction algorithms for volumes. But, in contrast to volumes, in order to obtain a smooth surface, the application of a *convolution* is required. The con-

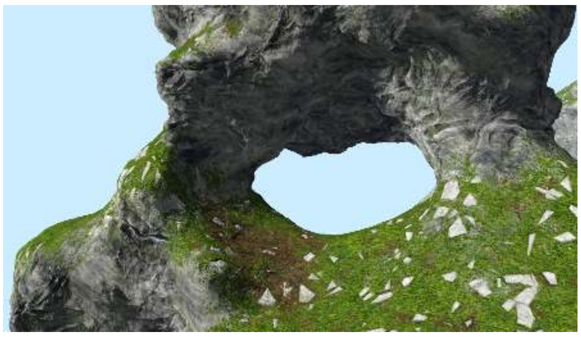
Figure 1: *Complex terrain feature (arch) rendered in real-time from a material stack representation with our approach.*

volution process is very expensive since it is necessary to determine the materials inside the (cubic) kernel support (cf. [PGMG09]). Due to layered representation, this is the major bottleneck for real-time rendering. In addition, usually rendering algorithms extract and render the surface at varying levels of detail piecewise. Apart from gaps between varying levels, the convolution produces discontinuities between the parts, even at the same level of detail.
To solve these problems we extend the work of [PGMG09] and [SDC09] and propose a novel concept for real-time rendering of stack-based terrains. Our aim is to extract a smooth surface to improve the visual fidelity. Our contribution can be summarized as follows:

- We introduce a novel method for constructing a level of detail hierarchy for material stacks. Since the aggregation of material stacks is not formally defined, the method is based on a heuristic (see Section 3.1).
- We develop an multistage quadrangulation pipeline, which transforms the stacks into a density function properly and extracts the surface in real-time using the dual contouring approach of [SDC09]. Furthermore, in contrast to the original approach the surface extraction is based on general purpose parallel hardware and allows for generating quadrilateral *indexed* meshes (see Section 3.2).
- We describe a stitching approach addressing the connectivity of both same and varying levels of detail. This way, the rendering guarantees a continuous terrain surface (see Section 3.4).

Our results (see Section 4) confirm the real-time capabilities of the proposed approach.

2. Related Work

Real-time rendering of terrain data needs to address three major aspects: data representation, surface rendering, and level of detail techniques.

Traditionally, terrain rendering relies on height-fields, a two dimensional regular grid of height values. Apart from the efficient procedural generation of terrains (cf. [SdKT*09]), this compact representation allows for rendering very large terrains (cf. [PG07]). In [PO06] multiply layers of height-fields are used to present non-height-field mesostructure details, whereas [CPO10] presents an efficient lossless compression technique for such multi layered displacement maps.
Material stacks are based on the layered data representation for height-fields proposed in [BF01]: At a grid point, different materials with certain properties are stacked. In [PGMG09] material stacks are used for modeling complex terrains. The special material air enables the modeling of complex terrain features as illustrated in Figure 2. In contrast to a voxel representation, material stacks are very resource-friendly.
A voxel representation implicitly describes arbitrary 3D surfaces by a 3D grid of density values: the *density function*.

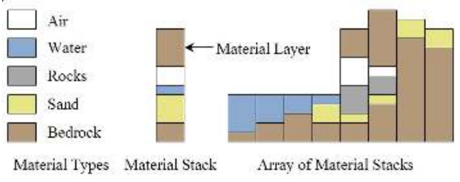

Figure 2: *Overview and example of the material stack data-structure from [PGMG09].*

In [Gei07] voxels are used to procedurally model complex large terrains. Whereas [BFO*07] use voxels to simulate the geological formation of Goblins.

For the rendering of the previously described data representation, the surface needs to be extracted. Usually, surface extraction is distinguished into ray-casting and polygonal surface extraction. Ray-casting casts rays to determine the visibility of surface points. For instance, [QQZ*03, DKW09, CNLE09, LK10] apply ray-casting for real-time rendering of large models, whereas [AGD10] combines ray-casting with polygonal rendering.
Polygonal surface extraction algorithms generate a mesh that approximates the implicit surface. For classic height-field rendering we refer to [PG07]. For 3D terrain data, methods of volume rendering can be applied. In [Gei07, GT07], for instance, a parallel implementation of the well-documented Marching Cube algorithm (cf. [NY06]) is proposed to extract the surface in real-time. For material stacks, in [PGMG09] the terrain surface is generated by Marching Cube as well. However, in order to generate a voxel representation, the discrete material stacks need to be convolved with a convolution kernel beforehand.
Along with the Marching Cube, the Dual Contouring [JLSW02] is another efficient algorithm. Dual Contouring generates a single vertex for all cells that intersect the surface. The vertex position is determined by minimizing the quadratic error function (QEF). [SDC09] propose several modifications that improve the performance including the adaption to regular grids, an efficient technique for face generation, and a particle-based error minimization. The face generation process relies on a look-up table, which we use in our approach as well.

There exists a wide range of level of detail techniques. Usually, techniques for 2D terrain rendering rely on quadtrees (e.g. [And07]) and bintrees (e.g. [BGP09]), respectively. For 3D terrain data similar data-structures can be applied. For instance, in [Gei07, GT07] an octree is used to manage 3D terrain volume. [LK10] rely on a sparse voxel octree and [CNLE09] use a combination of a N^3 tree and mipmapped 3D texture tiles to visualize arbitrary models at a high degree of geometric detail. For a comprehensive overview, we refer to [Dac06] and the excellent surveys of [Paj02, PG07, FKP05, DGY07].

Implications: Material stacks compactly store feature-rich 3D terrains. However, due to the required convolution and subsequent surface extraction, the rendering is very complex. Especially the surface extraction is a major cost factor. Ray-casting requires a voxel representation of the terrain and, hence, contradicts the compact representation. In contrast to that, for high quality meshes, polygonal surfaces are more memory efficient but require expensive computations. However, meshes are extracted once and can be reused for subsequent frames.
In conclusion, we expect that the real-time rendering of terrains relying on material stacks is feasible as long as the surface extraction can be realized in real-time.

3. Design of a Stack-based Terrain Renderer

In this Section we introduce our novel real-time rendering approach for stack-based terrain rendering. The work-flow for the presentation of 3D terrains can be outlined as follows:

Generation of the terrain data: There barely exist 3D terrain data-sets. To generate test data-sets we (a) adapt the procedural method of [Gei07] to material stacks and (b) develop an interactive editor for stack-based terrains.

Generation of a level of detail hierarchy: To reduce the rendering effort a level of detail structure is required. This is necessary as not all parts of the terrain need to be presented at the highest level of detail. Usually, level of detail hierarchies are computed using geometric properties. However, material stacks consist of materials. The aggregation of different materials is formally not defined. We propose a heuristic method for aggregate stacks. Due to the properties of material stacks, we can rely on a 2D based simplification instead of a 3D one. As a result, we obtain a compact MipMap representation of the stack-based terrain (see Section 3.1).

Rendering of terrain data: The process of rendering requires the extraction of the implicit surface. However, the implicit surface is described via convolution of stacks - this might cause problems: The convolution requires a determination of the material in the kernel volume, which is a very expensive calculation. To solve this problem, we transform the stacks into a density function. The density function is smoothed in a way that it represents the surface in the required quality. Thereafter, we extract the surface from the density function. Given comparatively good caching and rendering capabilities, we use a polygonal representation. Both the transformations and the extraction are organized in a pipeline. Major benefits are: First, a real-time surface extraction from stack-based terrain data and second, an indexed quadrilateral mesh. The pipeline is henceforth referred to as *quadrangulation pipeline* (see Section 3.2).

Considering the performance: Real-time rendering puts high demands on the performance. Thus, an efficient level of detail selection, an efficient culling, and a batched rendering approach are required. In order to fulfill these demands additional acceleration data-structures are necessary. Usually, terrain rendering relies on a 2D space partitioning. However, due to 3D data a stack-based terrain requires 3D space partitioning. We, therefore, apply an octree decomposition of the terrain: The nodes merely refer to $n \times n$ stacks at a corresponding level of detail in the terrain data (MipMap). This way, the octree accelerates the rendering, while the MipMap provides an efficient way to manage data. The octree is traverse and nodes are selected based on a distance metric. For nodes selected the surface mesh is generated by extracting the surface from the associated stacks. To take advantage of frame to frame coherence we cache surface meshes for subsequent frames in the octree node (see Section 3.3).

Considering the quality: The proposed rendering approach uses an octree to render the terrain piece by piece at varying levels of detail. To select appropriate nodes a level of detail metric is required. Our aim is to select nodes in such a way that the visual representations are of equal quality. We, thus, select nodes that nearly cover equal-sized regions in screen space. This approach, though, leads to another problem: Varying levels of detail introduce gaps and cracks. A continuous terrain surface without visual discontinuities is, however, desired. To overcome this obstacle, we apply a stitching approach. In contrast to height-field rendering, stitching needs to consider both the convolution and the topological variances between varying levels. We, therefore, introduce a method that (a) considers the convolution by extending the density function, (b) fits the topology by adjusting the density function, and (c) removes t-junctions by redirecting indexing of vertices (see Section 3.4).

3.1. LoD Hierarchy Construction

Using the initial-data we aim at generating multiple levels of detail for the rendering process (see Figure 4). The *quadtree decomposition* is one of the preferred methods for two dimensional regular grids. By taking the average of four adjacent samples from the previous level, we build a new level in the quadtree. Given the fact that stacks are organized on a regular grid, this method generates multiple levels of detail efficiently while simultaneously making full use of the stack-based representation. More precisely, each level k is a stack-based terrain approximating the original terrain with $2^k \times 2^k$ stacks, similar to a MipMap representation.
A problem, though, is the fact that usually aggregation is based on geometry properties. Stacks consist of different materials. It is formally neither defined how materials are aggregated nor how the average of different materials can be computed. The question is: How can stacks are combined in a ways that the new stack approximates its ancestors in the hierarchy to the most accurate extent?

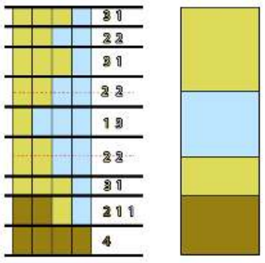

Figure 3: *Illustrate the aggregation of stacks. The four stacks (left) are subdivided into segments based on the number of covering material layers. The new stack (right) is constructed by analyzing the material composition.*

Our heuristic approach that relies on a piece wisely consideration of stacks yields to accurate results. For the aggregation the stacks are subdivided into segments. For each segment the material composition is analyzed and the material layer for the new stack is determined as follows:

1. A material is dominant in a segment. This dominant material is chosen for the segment.
2. No material is dominant. Instead two or more materials appear evenly. In this case, the material composition of the surrounding segments needs to be considered.

   a. If one of the materials in the surrounding segments is dominant this material is chosen for the segment.
   b. If one material is dominant in one direction and the other material in the opposite direction the segment is split and the material is distributed to the partial segments.
   c. If the material cannot be determined by (a) or (b) the number of considered segments is enlarged until a solution is found.

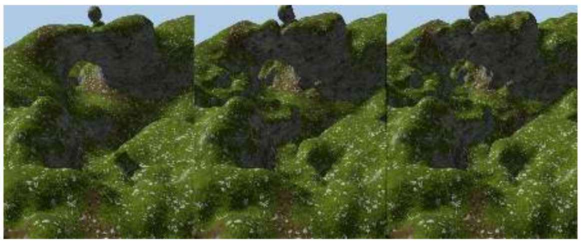

Figure 4: *Illustrates different levels of detail: (left) low, (middle) medium and (right) high level of detail.*

3.2. Quadrangulation Pipeline

The generation of the mesh is a critical process of stack-based terrain rendering. First, the quadrangulation needs to be performed on the fly which implies the need to meet real-time constraints. Second, a high quality and compact mesh is desired. Both require the extensive use of parallel hardware. However, to meet the second criteria it is necessary to break the parallel processing and to synchronize results. Due to expensive time costs, we target at synchronizing as few times as possible. To solve this difficulty we develop a multi-staged pipeline, whereby synchronization is only performed between stages. Consequently, stages can take full advantage of parallel processing. At the same time it is guaranteed that the data between stages is consistent. The pipeline is structured as follows (see Figure 5): First, the stack data is transferred to the GPU and transformed to a density function. Then, the density function is smoothed and subdivided into cells. Each cell is classified with regard to vertex and face generation. Subsequently, based on these classifications memory is allocated and vertices and normals as well as face indices for the mesh are computed. In the last step, the number of generated faces is read back to the CPU for the subsequent render calls.

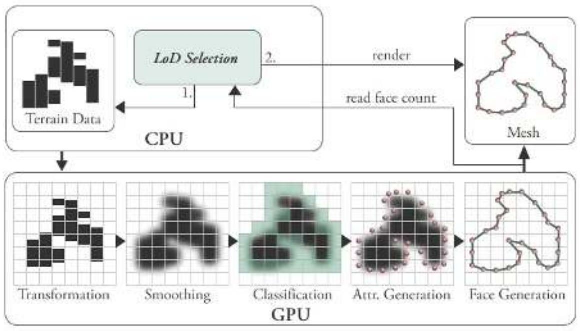

Figure 5: *Overview of the quadrangulation pipeline. The pipeline generates a surface mesh from stack-based terrain data entirely on GPU.*

3.2.1. Stacks to Density Transformation

The transformation stage converts the stacks into a combined density function by accumulating the individual material densities. The stacks should be converted in a way that the resulting density function characterizes the same surface. First of all, stacks are transferred to the hardware device memory (i.e. GPU). Afterwards, the density values are estimated by the coverage of the stacks and a 3D grid representing the combined density function. The procedure is similar to line rasterization. For each cell in the grid the density is defined by the percentaged coverage of the material layers. Material air is not considered.
To simplify the method, we predefine that one stack only effects a single column in the 3D grid. In other words, the ground planes of both the stacks and the grid have equal extents. Thus, stacks can be independently processed in a parallel manner.
For each stack, all layers are successively passed through.

Each layer L^i is characterized by a start height L^i_s, an end height L^i_e, and the material L^i_m. The algorithm can be summarized as follows:

- The index of the first covered cell is determine by flooring the start height of the layer: $floor(L^i_s)$. The density is derived from the fractional part of the start height: $1 - frac(L^i_s)$.
- The last covered cell is determined by flooring the end height of the layer: $floor(L^i_e)$. The density is equal to the fractional part of the end height: $frac(L^i_s)$.
- All cells between the first and the last cell are fully covered. Hence, density is maximal.

During the process of transformation two exceptions need to be considered: First, a cell is covered by more than one layer. In this case, the densities are added up. Second, the first and last cells are equal. In that case, the density is defined by $L^i_e - L^i_s$.

Optimizations: Under some circumstances, multiple materials are not required. This is the case when using procedural texturing techniques (cf. [Dac06]). In this situation, it is merely necessary to distinguish between air and non air material. The procedure remains the same as described above expect for the storage of material information.

3.2.2. Smoothing & Enhancing

After the stacks have been transformed to a density function, the density function needs to be smoothed. Stacks allow for a continuous description of the terrain in vertical direction. However, stacks are organized on a discrete grid. Thus, a cell in the density function covers a stack or not, resulting in a binary representation for two of three dimensions. This leads to aliasing artifacts appearing as block-shaped structures in the output image. To solve this problem, the density function is smoothed. We apply a separable low-pass filter (e.g. *box* or *gauss* filter) to the density function. The choice of filter size is a trade-off between losing to much detail, insufficient smoothing and performance.

3.2.3. Classification and Allocation

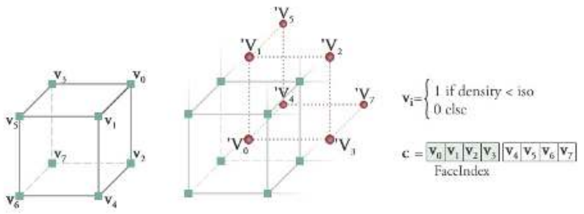
v_3
v_0
v_5
v_1
v_7
v_2
v_6
v_4
'V5
'V1
'V2
'V4
'V7
'V0
'V3
$v_i = \begin{cases} 1 \text{ if density} < \text{iso} \\ 0 \text{ else} \end{cases}$
c = v0 v1 v2 v3 v4 v5 v6 v7
FaceIndex

Figure 6: *Classification for face and vertex generation.*

For the surface extraction we use the dual contouring approach of [SDC09]. Therefore, with regard to face generation each cell needs to be classified. However, to avoid duplicate vertices as they usually appear in parallel approaches, we redirect face and vertex generation to subsequent stages. This requires, first, a classification for both face and vertex generation; second, a successive indexing of vertices; third, a unique mapping between cell and vertex indices; and last, a mapping between cells and offsets in the gapless face (index) buffer.
We solve the problem by means of a lookup table that is generated during the classification. This table provides information for subsequent stages and stores the classification $\mathbf{c}$, the vertex index $\mathbf{i_v}$ and the index buffer offset $\mathbf{o}$ of each cell. The major algorithm can be divided into two phases: classification and allocation, whereby allocation refers to the vertex index and the offset determination.

Classification: For determining the number of faces four density values are sufficient [SDC09]. If a cell generates one or more faces it needs to generate a vertex as well. However, a cell always generates a vertex if one or more density values exceed the iso-value. Hence, we consider the eight density values of a cell, similar to Marching Cube. The resulting classification is used as lookup index for both vertex and face generation (see Figure 6).

Allocation: The major problem is that we neither know how many vertices are created nor how many faces. In order to determine the vertex index $\mathbf{i_v}$ as well as the buffer offset $\mathbf{o}$, indices need to be accumulated by visiting cells successively. To minimize the massive synchronization overhead, we (a) locally determine the face as well as vertex count for each row simultaneous and (b) synchronously accumulate the counters (i.e. *atomic add*).

3.2.4. Attribute Generation

For all cells that contribute to the surface (that is for the classification $0 < \mathbf{c} < 255$), vertex attributes are computed. The attributes are stored in a corresponding buffer, whereby the location is determined by the vertex index $\mathbf{i_v}$ that is associated with the observed cell. Hence, each cell can be processed independently in parallel.

Position: The dual contouring approach generates vertex positions by minimizing the quadric error to the implicit surface (cf. [JLSW02]). However, the resulting sharp features contradict the smoothing requirements. Instead, we use the average of the vertices that are generated by the Marching Cube algorithm (cf. [SDC09]). This, additionally, smoothes the surface while simultaneously avoiding expensive computations.

Normal: For each vertex a normal can be derived directly from the density function f. The normal at a surface point is equal to the gradient direction which is defined by the partial derivation as follows: $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})$. We compute the partial derivation by applying a 3D sobel operator.

Material: The material composition for each vertex is determined by sampling the combined density functions (see Section 3.1). Due to the fact that the material ratio must always be 1, the resulting material vector is normalized for the rendering.

3.2.5. Face Generation & Read back

In the last step, each cell which contributes to the surface generates its faces and stores the indices in the index buffer. The location of the first index is given by the index buffer offset $\mathbf{o}$ associated with the cell. The number of faces is determined by the classification $\mathbf{c}$, whereby the faces can be derived from the lookup table proposed in [SDC09]. The lookup table merely defines the adjacent cells between which the faces are constructed. The *real* vertex index $\mathbf{i_v}$ for those cells is fetched from our lookup table (see Section 3.2.3).

In order to render the mesh the total number of faces needs to be read back to CPU. During the allocation we accumulate the total face count into a separate buffer which is read out after all stages have been finished.

3.3. Rendering Algorithm

For the rendering we need an acceleration data-structure that allows for efficient culling and level of detail selection. For the terrain data we use a quadtree decomposition. However, due to the following reasons, this data-structure is unfavorable for rendering: The visibility culling relies on the bounding volumes of the quadtree nodes. But in contrast to height fields, where only a single surface exists per grid point, material stacks may have multiple surfaces per grid point due to holes or caves. Because these surfaces might be at quite different heights, simple bounding-box-based visibility estimation is likely to be too conservative in many cases. Beyond this, as a measure for level of detail selection the distance calculation between viewer and nodes is insufficient. To solve this problem we use an octree decomposition. The octree nodes refer to stacks in the terrain data MipMap: A level in the octree corresponds to a level in the MipMap, whereas each node refers only to the stacks that are covered by the octree bounds (see Figure 7). If a node is rendered, a surface mesh is extracted which is based on the referred stacks. To exploit frame to frame coherence, the surface mesh is cached for subsequent frames. Consequently, with the use of two data-structures, we preserve the advantages of material stacks for the data management while simultaneously improving the culling and level of detail selection for the rendering.

To compute the approximation quality we rely on a distance base metric. We estimate the screen space extend ε of a node by using the following equation for the usual perspective projection (cf. [LP02]):

$$\varepsilon = \delta_i \frac{w}{2d_i \tan\frac{\phi}{2}} \tag{1}$$

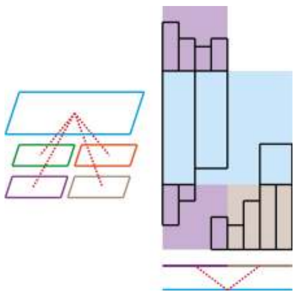

Figure 7: *Illustration of the interrelation between the quadtree (left) and the octree decomposition (right). Each octree node visualizes the terrain based on the stacks that correspond to the color-highlighted quadtree nodes.*

δ_i is the size of the node i in object space, d_i is the closest distance between the node i and the view-point; w is the number of pixels on ϕ, which is the field of view. The resulting screen space extent ε is compared to a user defined quality threshold τ. Nodes are split as long as $\varepsilon > \tau$ and hence, as long as they do not meet the desired approximation quality.

3.4. Quality Improvement: Stitching

Due to the application of a level of detail metric the terrain surface is piecewise approximated (see Section 3.3). This leads to visual artifacts between adjacent nodes, which appear as cracks and gaps. In order to guarantee a continuous surface the attributes of adjacent border vertices need to be equal. We derive the vertex attributes from a density function (see Section 3.2.4). Hence, stitching can be reduced to compute vertex attributes from equal density values. We distinguish (a) stitching between adjacent nodes with the same level of detail and (b) stitching between adjacent nodes with varying levels of detail.

3.4.1. Same Level of Detail

To obtain a continuous surface, the volumes of adjacent nodes need to overlap by a single cell. In this case, adjacent border vertices are coincident. However, due to a lack of adjacent information the smoothing (cf. Section 3.2.2) yields to different density values. To solve this problem, we extend the density function by half of the convolution filter size.

3.4.2. Varying Level of Detail

Due to the fact that nodes with a lower level of detail might be adjacent to two or more nodes with a higher level of detail, the stitching between varying levels requires more effort. Furthermore, topology can be changed between the different levels of detail.

To deal with this problem we appoint the following preconditions: Levels of detail between adjacent nodes can only vary by one level (as it is usual for terrain rendering). Hence, sampling rate of the density functions differs by factor 2. Moreover, the border of the node with the higher level of detail K_n is fitted to the node with a lower level of detail K_{n-1}. Our basic idea is to modify the density values of the higher detailed border of K_n in order to fit the lower detailed border of K_{n-1}. During the attribute generation, T-junctions occurring due to the twice sampling rate are removed. The algorithm is illustrated in Figure 8 and can be outlined as follows:

Topology fitting: To fit the topology of the border of K_n, we replace the density values $d^n_{i,j,k}$ with the corresponding density values of the previous level $d^{n-1}_{i/2,j/2,k/2}$ via point sampling. This replacement is proceeded for all density values that influence the attribute generation of the border vertices. Consequently, the density functions in the overlapping region of K_n and K_{n-1} are equal.

T-junction removal: Due to the different sampling rates of the generated meshes, T-junctions occur at the boundary. To remove unnecessary vertices, we modify the vertex lookup of the border vertices. Therefore, odd components of a cell lookup vector are round up to the next higher even value. As a result, vertices are snapped together as illustrated in Figure 8. This introduces degenerated faces. However, neither does this affect the performance nor does this affect the visual quality [HSH09].

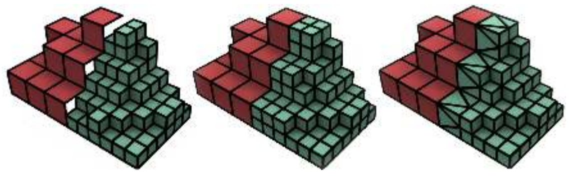
Figure 8: *Illustration of the stitching process. Varying levels of detail (left) are stitched by fitting the topology of the higher detailed node (middle) followed by the removal of T-junctions (right).*

4. Implementation and Results

The rendering concept has been prototypically implemented in C++ and DirectX11 (see Figure 9). The quadrangulation pipeline has been implemented entirely on the GPU. Each stage is realized as *Compute Shader* [Boy08]. The quadrilateral mesh is rendered as triangle mesh. Near the camera we rely on hardware tessellation for per vertex displacement mapping.

The tested data-set has been generated with our terrain editor (see Section 3). The data-set has an extent of $4km \times 8km$ with a maximum height of $768m$ and is comparable to a volume representation of $2048 \times 4096 \times 512$ voxels. The MipMap representation of the terrain requires approximately $106MB$. In contrast to that, a corresponding voxel representation would require more that 20GB (uncompressed).

The performance statistics has been captured over several thousands of frames at full HD (1920×1080) resolution and has been performed on an Acer Espire M7811 (Intel I7 860) equipped with Radeon 5870. We use a node size of 17^3 and a cell screen space size of $10px$. We render approximately 435600 quadrilateral patches at 56.2 frames per second. To estimate the different costs, we disable the tessellation and achieve an average frame rate of 192.8 frames per second. We found out that the average frame rate is limited by recurrent node quadrangulation. The time cost for quadrangulation of a single node takes approximately 1.4ms (GPUPerfStudio). In case a node is split or merged, several quadrangulations need to be carried out. Hence, the duration of this frame might exceed 40ms and real-time is not achieved. We solved this problem in the following way:

In each frame we can update ≈ 40 nodes to achieve 25Hz. In worst case, in an octree the split of nodes requires the update of 8 nodes, while the merging requires the update of the merged node and of 24 adjacent nodes. However, due to the distance based LoD metric less than the half of adjacent nodes might be updated. By amortizing the split and merge of nodes over several frames, a frame rate above 25Hz can be achieved. This has drawbacks though: Whenever the viewer moves too fast, the amortization strategy leads to popping artifacts and a delayed LoD switch. By incrementing the screen space error tolerance less nodes need to be updated and the viewer can move faster. However, due to the LoD selection popping artifacts still occur.

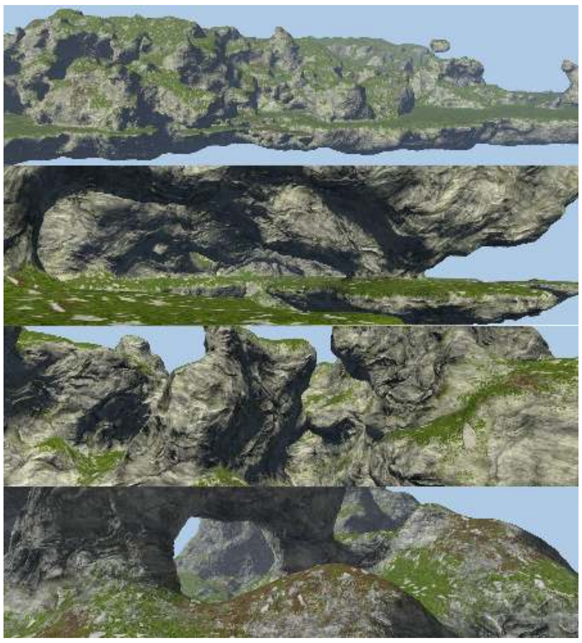
Figure 9: *Screen dumps from the prototypical implementation and the tested data-set.*

5. Conclusion

We present a novel approach for real-time rendering of 3D terrains. The terrain representation relies on material stacks. The material stacks compactly represent complex 3D terrain data (e.g. arches or caves). With our approach such data can be visualized directly and provides new options for entertainment and modeling applications.

Prior to rendering, a MipMap representation of the terrain data is generated. For the required aggregation of materials, we introduce a heuristic approach. The MipMap representation is compact and allows for managing relatively large terrains without out of core strategies. Due to the 3D characteristics of the data, this representation is inappropriate for rendering. Hence, the rendering relies on an octree decomposition of the terrain area, whereby the octree nodes refer merely to the terrain data in the MipMap. During the rendering, we extract the terrain surface from the material stacks in real-time. The surface extraction is designed as pipeline that takes the terrain data as input and generates an indexed quadrilateral mesh as output. In order to guarantee a continuous surface the stitching between various levels of detail is considered. Our results support the real-time capabilities of our approach.

Our future work will focus on the improvement of the pipeline's different stages and the implementation of a more sufficient amortization strategy. The scope of future work includes (1) a seamless combination of our work and classical height-field rendering to decrease the surface extraction effort; (2) a geomorphing approach to avoid popping artifacts during LoD switches; and (3) the inclusion of adjacent information during surface extraction in order to generate real smooth surfaces (tessellation hardware).

References

[AGD10] Ammann L., Génevaux O., Dischler J.: Hybrid rendering of dynamic heightfields using ray-casting and mesh rasterization. In *Proceedings of Graphics Interface* (2010), Canadian Information Processing Society, pp. 161–168. 2

[And07] Andersson J.: Terrain rendering in frostbite using procedural shader splatting. In *ACM SIGGRAPH courses* (2007), ACM, pp. 38–58. 2

[BF01] Benes B., Forsbach R.: Layered data representation for visual simulation of terrain erosion. In *Computer Graphics, Spring Conference* (2001), IEEE, pp. 80–86. 1, 2

[BFO*07] Beardall M., Farley M., Ouderkirk D., Smith J., Jones M., Egbert P.: Goblins by spheroidal weathering. In *Eurographics workshop on natural phenomena* (2007), pp. 7–14. 2

[BGP09] Bösch J., Goswami P., Pajarola R.: RASTeR: Simple and efficient terrain rendering on the GPU. *Proceedings of Eurographics Areas Papers* (2009), 35–42. 2

[Boy08] Boyd C.: The DirectX 11 Compute Shader. *ACM SIGGRAPH classes* (2008). 7

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2009), ACM Press. 2

[CPO10] Chun Y., Park S., Oh K.: Multiple layer displacement mapping with lossless image compression. *Entertainment for Education. Digital Techniques and Systems* (2010), 518–528. 2

[Dac06] Dachsbacher C.: *Interactive Terrain Rendering: Towards Realism with Procedural Models and Graphics Hardware*. PhD thesis, Universität Erlangen-Nürnberg, Universitätsstraße. 4, 91054 Erlangen, 2006. 2, 5

[DGY07] Dietrich A., Gobbetti E., Yoon S.: Massive-Model Rendering Techniques. *IEEE Computer Graphics and Applications* (2007), 20–34. 2

[DKW09] Dick C., Krüger J., Westermann R.: Gpu raycasting for scalable terrain rendering. *Proceedings of Eurographics Areas Papers* (2009), 43–50. 2

[FKP05] Floriani L., Kobbelt L., Puppo E.: A survey on data structures for level-of-detail models. *Advances in multiresolution for geometric modelling* (2005), 49–74. 2

[Gei07] Geiss R.: *GPU Gems 3: Generating Complex Procedural Terrains Using the GPU*. Addison-Wesley, 2007, pp. 7–37. 2, 3

[GT07] Geiss R., Thompson M.: NVIDIA Demo Team Secrets Cascades. Presentation at Game Developers Conference, 2007. 2

[HSH09] Hu L., Sander P., Hoppe H.: Parallel view-dependent refinement of progressive meshes. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2009), ACM New York, NY, USA, pp. 169–176. 7

[JLSW02] Ju T., Losasso F., Schaefer S., Warren J.: Dual contouring of hermite data. *ACM Transactions on Graphics 21*, 3 (2002), 339–346. 2, 5

[LK10] Laine S., Karras T.: Efficient Sparse Voxel Octrees–Analysis, Extensions, and Implementation. *NVIDIA Corporation* (2010). 2

[LP02] Lindstrom P., Pascucci V.: Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 239–254. 6

[NY06] Newman T., Yi H.: A survey of the marching cubes algorithm. *Computers & Graphics 30*, 5 (2006), 854–879. 2

[Paj02] Pajarola R.: Overview of quadtree-based terrain triangulation and visualization. *Technical Report* (2002). 2

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

[PGMG09] Peytavie A., Galin E., Merillou S., Grosjean J.: Arches: a Framework for Modeling Complex Terrains. *Proceedings of Eurographics 28*, 2 (2009), 457–467. 1, 2

[PO06] Policarpo F., Oliveira M.: Relief mapping of non-height-field surface details. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2006), ACM, pp. 55–62. 2

[QQZ*03] Qu H., Qiu F., Zhang N., Kaufman A., Wan M.: Ray tracing height fields. In *Computer Graphics International* (2003), IEEE, pp. 202–207. 2

[SDC09] Schmitz L., Dietrich C., Comba J.: Efficient and High Quality Contouring of Isosurfaces on Uniform Grids. In *Computer Graphics and Image Processing* (2009), IEEE, pp. 64–71. 2, 5, 6

[SdKT*09] Smelik R., de Kraker K., Tutenel T., Bidarra R., Groenewegen S.: A survey of procedural methods for terrain modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation* (2009). 2