GPU-based Ray Casting of Stacked Out-of-Core Height Fields

Christopher Lux Bernd Fröhlich

Bauhaus-Universität Weimar

Abstract. We developed a ray casting-based rendering system for the visualization of geological subsurface models consisting of multiple highly detailed height fields. Based on a shared out-of-core data management system, we virtualize the access to the height fields, allowing us to treat the individual surfaces at different local levels of detail. The visualization of an entire stack of height-field surfaces is accomplished in a single rendering pass using a two-level acceleration structure for efficient ray intersection computations. This structure combines a minimum-maximum quadtree for empty-space skipping and a sorted list of depth intervals to restrict ray intersection searches to relevant height fields and depth ranges. We demonstrate that our system is able to render multiple height fields consisting of hundreds of millions of points in real-time.

1 Introduction

The oil and gas industry is continuously improving the seismic coverage of subsurface regions in existing and newly developed oil fields. During this process, very large volumetric seismic surveys are generated using the principles of reflection seismology. The resulting volumetric data sets represent the magnitude of seismic wave-reflections in the earth's subsurface. Geologists and geophysicists use these seismic volumes to create a geological model of the most important subsurface structures in order to identify potential hydrocarbon reservoirs. Horizons are a fundamental part of the geological model representing the interface between layers of different materials in the ground. The ever increasing size of seismic surveys generates extremely large horizon geometries composed of hundreds of millions of points (Figure 1). Traditional rasterization methods cannot render such data sets in real-time. While horizon surfaces are commonly represented as height fields, general terrain rendering approaches [1] have not been adapted to deal with the specific structure of multiple horizon layers.

We developed an efficient ray casting-based rendering system for the out-of-core visualization of sets of large stacked horizons. Our method employs a multi-resolution data representation and makes use of a minimum-maximum quadtree over the tiled horizon height fields to speed up the ray traversal. Sorted height intervals in the quadtree cells restrict the intersection searches to relevant horizons and depth ranges. We virtualize the access to the underlying horizon height fields such that each horizon can be locally treated at different levels of

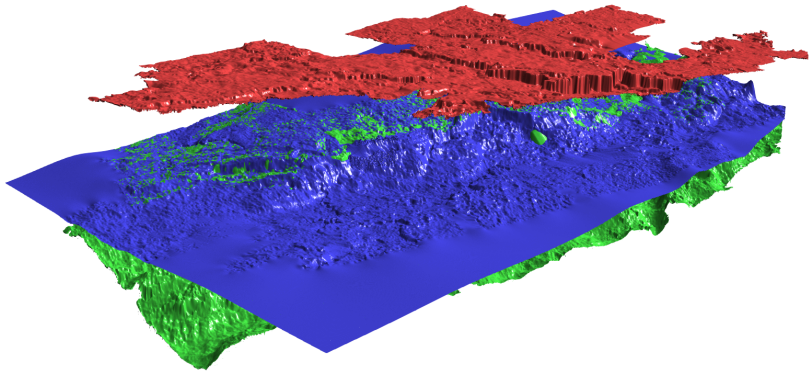


Fig. 1: This figure shows three different horizons extracted from a common seismic survey. While the upper most horizon (red) is spatially isolated, the lower horizons (green and blue) are partially overlapping. The original resolution of each horizon is 6870×14300 points.

detail (e. g. occluded horizon parts are represented at a much lower resolution). Our out-of-core data management system supports geological models consisting of multiple large height-field data sets, potentially exceeding the size of the graphics memory or even the main memory. A feedback mechanism is employed during rendering which directly generates level-of-detail information for updating the cut through the multi-resolution hierarchies on a frame-to-frame basis.

Our work is motivated by the observation that with increasing screen resolutions and screen-space errors below one pixel, the geometry throughput of current GPUs is becoming a major performance bottleneck and ray casting techniques can provide comparable performance for large data sets [2, 3]. However, none of the existing ray casting-based rendering systems are capable to efficiently visualize complete stacks of highly detailed, mutually occluding and overlapping horizons contained in geological models.

The main contributions of our out-of-core stacked height-field ray casting system include:

- A two-level acceleration structure for fast ray traversal and efficient ray intersection computations with stacked height fields.
- A single-pass rendering approach, which also generates level-of-detail feedback for updating the cuts through the multi-resolution representations of the height fields.
- A two-level out-of-core data management system, which provides virtualized access to multi-resolution height-field data.

Our implementation demonstrates that we can render multiple horizons consisting of hundreds of millions of points in real-time on a single GPU. The ray casting approach also facilitates the integration with volume ray casting, a highly desirable property in visualization systems for subsurface data.

2 Related Work

The direct visualization of height-field surfaces using ray casting-based methods is a very active and well explored field of computer graphics research. Early CPU-based methods, primarily targeted at terrain rendering applications, were based on a 2D-line rasterization of the projection of the ray into the two-dimensional height-field domain to determine the cells relevant for the actual intersection tests [4–7]. A hierarchical ray casting approach based on a pyramidal data structure was proposed by Cohen and Shaked [5]. They accelerate the ray traversal by employing a maximum-quadtree, storing the maximum height-displacement value of areas covered by its nodes in order to identify larger portions of the height field not intersected by the ray.

Enabled by the rapid evolution of powerful programmable graphics hardware, texture-based ray casting methods implemented directly on the GPU were introduced. The first published approach by Qu et al. [7] still uses a line rasterization approach similar to the traditional CPU-based approaches. The relief-mapping [8] methods pioneered by Policarpo et al. [9, 10] employ a parametric ray description combined with an initial uniform linear search which is refined by an eventual binary search restricted to the found intersection interval. Considering that the initial uniform stepping along the ray may miss high-frequency details in the height field, these methods are considered approximate. While these initially published GPU-based ray casting algorithms are not utilizing any kind of acceleration structures, later publications proposed different approaches. Donnelly [11] described the use of distance functions encoded in 3D-textures for empty-space skipping. The drastically increased texture memory requirements make this technique infeasible for the visualization of large horizon height fields. Later methods proposed by Dummer [12] and Policarpo et al. [10] exhibit drastically reduced memory requirements. They calculate cone ratios for each cell to describe empty space regions above the height field allowing fast search convergence during the ray traversal. The very high pre-computation times of these techniques only allow for the handling of quite small height-field data sets. This problem is addressed in the subsequent publications by Oh et al. [13] and Tevs et al. [14]. They build upon the traversal of a maximum-quadtree data structure on the GPU akin to the algorithm presented by Cohen and Shaked [5]. The idea of encoding the quadtree in the mipmap hierarchy of the height field results in very moderate memory requirements.

Visualizing a large height-field data set requires the use of level-of-detail and multi-resolution techniques to balance between rendering speed and memory requirements. Hierarchical multi-resolution data representations are traditionally applied to the visualization of large volumetric data sets [15]. Dick et al. [3] are able to handle arbitrarily large terrain data sets employing a multi-resolution quadtree representation of the tiled terrain elevation map. The virtualization of multi-resolution data representations enables the implementation of rendering algorithms, which can be implemented such that they are mostly unaware of the underlying multi-resolution representation of the data set. Kraus and Ertl [16] describe how to use a texture atlas to store individual image sub-tiles in a single

texture resource. They use an index texture for the translation of the spatial data sampling coordinates to the texture atlas cell containing the corresponding data. Generalized out-of-core texture data virtualization methods were introduced as a result [17].

Our multi-resolution height-field data virtualization is based on a quadtree representation of the height-field data sets. The selected quadtree cuts are stored in a 2D-texture atlas. Additionally, we are using a compact serialization of the quadtree cut similar to [18] for the translation of the virtual texture sampling coordinates to the texture atlas cell containing the corresponding data. Furthermore, to update the levels of detail represented through the multi-resolution hierarchies, we employ a feedback mechanism during rendering similar to the mechanism described by Hollemeersch et al. [19]. They require a separate rendering pass to write the required level-of-detail feedback information to a lower resolution target for fast evaluation. Using costly rendering approaches such as ray casting, this would involve a serious rendering overhead. Our system generates the feedback information directly during rendering while still allowing to sub-sample the actual screen resolution.

Policarpo et al. [10] introduced a method for handling a fixed number of height fields without any acceleration structure or virtualization in a single rendering pass. They simply move along the ray using a fixed sampling step and intersect all the height fields at once using vector operations on the GPU, which works efficiently for at most four height fields encoded in a single texture resource. In contrast, we handle each height-field layer as an individual data resource. This allows us to represent different parts of a horizon surface at different local levels of detail (e. g. occluded parts on individual horizons are represented at much lower resolution). Furthermore, we employ a minimum-maximum quadtree over the tiled horizon height fields to speed up the ray traversal and use sorted intersection intervals for the individual horizons to restrict the actual intersection searches.

3 Out-of-Core Data Virtualization

This section describes our out-of-core data virtualization and rendering system. We begin with a brief overview of the basic system architecture and the relationships of the most important components followed by a more detailed description of our resource management and level-of-detail feedback mechanism.

3.1 System Architecture

The foundation of our system is an efficient out-of-core texture management system which is designed to enable the handling of geological models consisting of multiple large horizons. The height fields describing the horizon surfaces are managed as two-dimensional single-channel texture resources by our system. The main parts of the system include the page cache, the page atlas and the level-of-detail feedback mechanism (Figure 2).

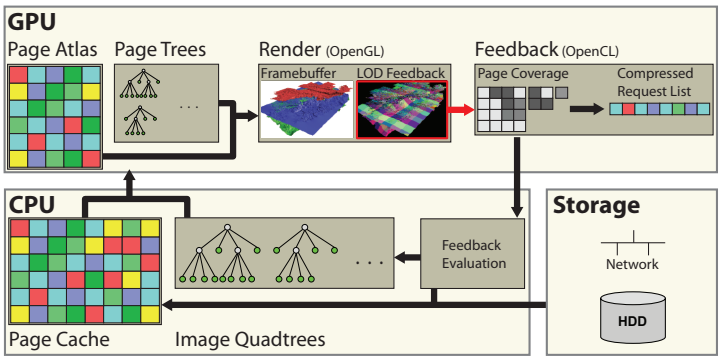
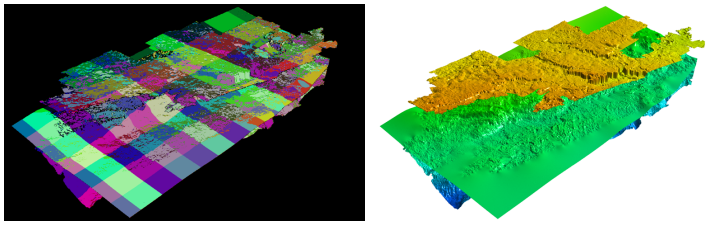


Fig. 2: This figures shows an overview of the out-of-core data virtualization and rendering system. Two large memory resources are maintained, one on the CPU and one on the GPU - the page cache and page atlas. The image data is maintained as quadtree representations using a compact serialization scheme on the GPU to provide efficient translations of virtual to physical texture coordinates in the page atlas.

Visualizing height-field data potentially exceeding available graphics and system memory resources requires the use of level-of-detail and multi-resolution techniques to balance between visualization quality and memory requirements. Hierarchical multi-resolution data representations are traditionally applied for the visualization of large volumetric data sets [15]. Similar to these approaches, we use a quadtree as the underlying data structure for the multi-resolution representation of the two-dimensional height-field data. All nodes in the quadtree are represented by tiles or pages of the same fixed size, which act as the basic paging unit throughout our memory management system.

For each height field we compute and continuously update a cut through its multi-resolution hierarchy using feedback information gathered during rendering (Section 3.2). The leaf nodes of these cuts define the actual working set of height-field pages on the GPU. The cut updates are incrementally performed using a greedy-style split-and-collapse algorithm, which considers a fixed texture memory budget [20]. During the update operation, only data currently resident in the main memory page cache is used and unavailable pages are requested to be asynchronously loaded and decoded from the out-of-core page pool. This approach prevents stalling of the update and rendering process due to slow transfers from external page sources.

The height-field data is accessed during rendering through two resources on the GPU: a single shared large page atlas texture of a fixed size containing the pages of the current working sets of all height fields, and a set of small textures representing a serialization of the current quadtree cuts for each height field. The quadtree cuts are used for the translation of virtual texture coordinates to the physical sampling location in the shared page atlas.



(a) Page-ID Feedback Buffer (b) Rendering Result

Fig. 3: This figure shows the results of the level-of-detail feedback mechanism during rendering. (a) Visualization of the determined page identifiers. (b) Final rendering based on our height-field ray casting.

3.2 Level-of-Detail Feedback Generation

Two basic approaches exist for generating view-dependent information concerning the required levels of detail of a multi-resolution representation: analytical methods and direct feedback mechanisms. Analytical methods try to determine the required levels of detail through view-dependent or data-dependent heuristics. These methods usually exhibit problems to consider occlusions in the data sets without the application of sophisticated occlusion culling techniques [21]. In contrast, the direct feedback methods typically employ additional rendering passes to generate information about required level of details for the current view in off-screen buffers. After transferring these buffers to the system memory, their contents are analyzed and the derived information is used to inform the respective multi-resolution update methods. These additional rendering passes usually use only small off-screen buffers, which are a fraction of the size of the actual view port, for minimizing read back latency and processing time.

Our system employs a direct screen-space feedback mechanism similar to Hollemeersch et al. [19]. The mechanism works in three stages. First, during rendering, identifiers for the actual required pages are determined for each pixel and saved into an off-screen buffer (Figure 3). Then, this buffer is used in an evaluation step directly on the GPU to generate a list of required pages and their respective pixel coverage. Finally, this compact list is transferred to the CPU where the coverage information of the pages is used to prioritize the nodes of the quadtrees. The page identifiers written to the feedback buffer encode the actual page position in the quadtree and in a height-field instance. The advantage of this approach is that a large part of the feedback buffer evaluation is performed in parallel directly on the GPU and the condensed list is generally orders of magnitudes smaller than the actual feedback buffer allowing for fast transfers to the system memory.

The computation of the level-of-detail information during rendering virtualized height fields is a straight forward process. However, the high run-time complexity of the height-field ray casting makes a separate rendering pass infeasible for feedback generation. For this reason, we generate the feedback information

directly during rendering. Usually this requires that the feedback buffer is bound as an additional rendering target to the current frame buffer (a multiple render target (MRT) setup). This would force the feedback buffer to be the same size as the used view port. Using direct texture image access functionality provided by current graphics hardware[1], we are able to sub-sample the view port by directly writing the feedback information to a lower resolution off-screen buffer, thereby avoiding the traditional MRT setup.

3.3 Height-Field Virtualization

Data virtualization refers to the abstraction of logical texture resources from the underlying data structures, effectively hiding the physical characteristics of the chosen memory layout. Our system stores the physical height-field tiles in the page atlas in graphics memory. In addition, we store a compact serialization of the quadtree cuts and their parent nodes up to the root node for virtual texture coordinate translation. The encoding of the quadtrees is similar to the octree encoding proposed by Lefebvre et al. [18]. Each node in this data structure holds information regarding where the corresponding page is located in the page atlas, including scaling information and child node links for inner nodes. This allows us to choose every level of detail currently available in the associated quadtree cut.

Translating a virtual texture coordinate into a physical sample coordinate in the page atlas involves the following steps: first, the required level of detail is determined for the requested sampling location; then, the quadtree is traversed from the root node down to the appropriate level resulting in the indirection information required to transform the initial virtual sampling position to the physical sampling position in the page atlas. If the appropriate level of detail is not available in the quadtree cut, the traversal stops at the highest currently available level and thus returns a lower resolution version of the requested page. As an extension to this approach, tri-linear data filtering is implemented by storing the upper and lower bound of the required level of detail for a virtual sampling position during the quadtree traversal and filtering the two resulting image samples accordingly. This way, aliasing artifacts can be effectively prevented.

Using the quadtree serialization method for virtual texture coordinate translation requires a tree traversal, but maintains a very small memory footprint per height-field data set. Although the traversal routine is benefiting from a good texture cache performance due to the small size and locality of the quadtree encoding, the cost of the tree traversal is not negligible. However, each ray typically samples the same height field tile multiple times and thus caching traversal information significantly reduces average traversal costs.

[1] This functionality is provided through the OpenGL4 EXT_shader_image_load_store extension.

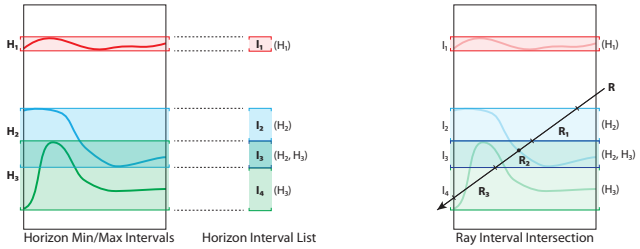


Fig. 4: This figure shows the construction and traversal of the sorted depth interval list for a single height-field tile. The left image shows the original minimum-maximum intervals for three horizons. The sorted interval list with associated horizons is derived from the intersections of the source intervals. The right image shows the ray-intersection search in the successive intervals, resulting in the found intersection of the ray interval R_2 in the interval I_3.

4 Ray Casting Stacked Height Fields

Horizon surfaces are derived from a common seismic volume. Thus, the entire stack of horizon surfaces can be defined as equally sized height fields within the lateral (x-y) domain of the volume. The access to each height field is virtualized using our texture virtualization system, which allows us to access the height values of different height fields at different levels of detail using the same global texture coordinates.

Our rendering approach is based on ray casting. We use a two-level acceleration structure for efficient intersections of a ray with a set of stacked height fields. A global minimum-maximum quadtree represents the primary structure for fast empty space skipping and a sorted list of height intervals is associated with each quadtree node to restrict intersection searches to relevant horizons and depth ranges. We restrict the size of the resulting quadtree data structure by building the minimum-maximum hierarchy based on tiled horizon height fields. In fact, only a cut through this global quadtree is required during rendering, which is a union of all the multi-resolution cuts of the individual height fields available on the GPU. The minimum-maximum ranges associated with each node are generated as the union of the individual ranges of the distinct height fields. However, we generate a sorted disjoint interval list for each quadtree node, which associates one or more horizons with each interval as shown in Figure 4.

The ray traversal of the minimum-maximum quadtree is performed in top-down order in a similar way as described by Oh et al. and Tevs et al. [13, 14]. The minimum-maximum intervals associated with each quadtree node are used to quickly discard nodes without possible intersections. When reaching a leaf node in the current cut through the global quadtree, the actual horizons contained in the sorted horizon interval list are intersected. The evaluation order of the interval list is dependent on the actual ray direction. If the ray is ascending in the z-direction, the list is evaluated front-to-back starting with the

interval with the smallest minimum value, while it is evaluated back-to-front for descending rays. If an intersection is found during the evaluation of an interval, it represents the closest height-field intersection and the evaluation process can be terminated early. In case of intervals associated with multiple horizons, we successively search all contained horizons for intersections in the appropriate order (Figure 4). Once a horizon intersection is found, we use the intersection point to restrict the subsequent searches in the remaining horizons associated with the interval. Due to the fact that the quadtree nodes represent horizon tiles containing e. g. 128×128 height values, we perform a linear search along the ray within a tile to find a potential intersection interval followed by a binary search for finding the actual intersection point.

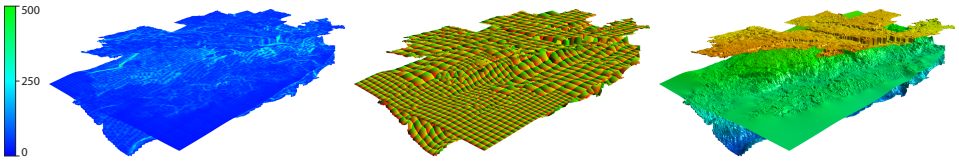
Even though there is no restriction on the tile size used for the construction of the global minimum-maximum quadtree, using a tile size less than or equal to the tile size utilized by the virtualization of the height fields allows for optimizations during the intersection search. Therefore, only a single query for the page atlas location of an associated height-field tile is required considering that all data lookups during the intersection search in a tile can be directly taken from this single atlas page. Thus, traversal costs in the serialized height-field cuts for localizing the atlas page are amortized over a complete search interval, thereby significantly reducing the overhead of the virtualization approach.

5 Results

We implemented the described rendering system using C++. OpenGL4 and GLSL were used for the rendering related aspects and OpenCL for the level-of-detail feedback evaluation, which is also generated directly on the GPU. All tests were performed on a 2.8 GHz Intel Core i7 workstation with 8 GiB RAM equipped with a single NVIDIA GeForce GTX 480 graphics board running Windows 7 x64.

We tested our system with a data set containing a stack of three partly overlapping horizons provided to us by a member of the oil and gas industry for public display (Figure 5). The dimensions of each height field defining a horizon surface are 6870×14300 points using 16 bit resolution per sample. The chosen page size for the virtualization of the height fields was 128^2, the page atlas size was 128 MiB and the page cache size was restricted to 1 GiB. The rendering tests were performed using a view port resolution of 1680×1050.

Our system is able to render scenes as shown in Figure 5 with interactive frame rates ranging from 20 Hz to 40 Hz depending on the viewer position and zoom level. We found that the rendering performance of our ray casting system is mainly dependent on the screen projection size of the height fields and the chosen sampling rate. The memory transfers between the shared resources of the data virtualization system have limited influence on the rendering performance. The level-of-detail feedback evaluation on the GPU introduces a small processing overhead below 0.5ms per rendering frame in our testing environment. We chose a feedback buffer size a quarter of the actual view port resolution (420×262).



(a) Iteration Steps (b) Level of Detail (c) Final Result

Fig. 5: Horizon stack containing three partially intersecting surfaces. The resolution of the underlying height fields is 6870×14300 points. The left image displays the number of iteration steps required to find the final intersection, with brighter colors representing larger iteration counts. The middle image shows the quadtree cuts selected during the level-of-detail evaluation. The left image shows the final rendering result using a default color map encoding the sample depth in the generating seismic survey.

We evaluated various tile sizes for the construction of the global minimum-maximum quadtree ranging from 32^2 to 128^2. The smaller tile sizes exhibited small improvements of 3-5% in rendering performance under shallow viewing angles compared to the largest size. However, under steep viewing angles, the introduced traversal overhead of the larger quadtree data structure resulted in a slight degradation of performance of 5-8%. The larger potential for empty space skipping using smaller tile sizes is canceled out by the complexity introduced by the depth interval list evaluation and the final intersection search in the height-field tiles. We found that using the same tile size for the global minimum-maximum quadtree as for the data virtualization results in the best compromise between data structure sizes on the GPU and rendering performance. Figure 5a shows that higher iteration counts occur in areas where the ray closely misses one horizon tile and intersects another one using a 128^2 tile resolution. Thus on the finest level, a ray needs to take an average of about 64 steps to find an intersection inside a tile assuming that such an intersection exists.

We also experimented with binary searches for finding the horizon intervals in a quadtree node. However, the limited number of actual depth intervals in our current data sets made this approach slower than simply searching linearly for the first relevant interval. For a larger number of stacked horizons, the binary search for the first relevant interval should improve performance.

Considering the caching of traversal information for the height-field virtualization mechanism during the ray-intersection search in a height-field tile, we found that an implicit caching mechanism implemented for the virtual data look ups results in better run-time performance than an explicit mechanism transforming the ray intersection search to the local coordinate system of the height-field tile in the page atlas. This allows for a much clearer implementation of the ray casting algorithm decoupled from the actual multi-resolution data representation.

6 Conclusions and Future Work

We presented a GPU-based ray casting system for the visualization of large stacked height-field data sets. Based on a shared out-of-core data management system, we virtualize the access to the height fields, allowing us to treat the individual surfaces at different local levels of detail. The multi-resolution data representations are updated using level-of-detail feedback information gathered directly during rendering. This provides a straightforward way to resolve occlusions between distinct surfaces without requiring additional occlusion culling techniques. The visualization of entire stacks of height-field surfaces is accomplished in a single rendering pass using a two-level acceleration structure for efficient ray intersection searches. This structure combines a minimum-maximum quadtree for empty-space skipping and a sorted list of depth intervals to restrict ray intersection searches to relevant height fields and depth ranges. The implementation shows that stacks of large height fields can be handled at interactive frame rates without loss of visual fidelity and moderate memory requirements.

The feedback information used to guide the update of the multi-resolution representations is currently based on a purely texture space level-of-detail metric. A combination of this approach with screen-space or data-based error metrics for the tile-based level-of-detail estimation can further improve rendering quality.

Our ultimate goal is a visualization system for subsurface data capable of interactively visualizing entire geological models. A highly desirable feature of such a system is the combined rendering of surface geometries and volume data. Typical geological models in the oil and gas domain can consist of a large number of highly detailed horizon surfaces and extremely large volume data sets. Currently no infrastructure exists for the efficient out-of-core management and rendering of geometry and volume data. Our ray casting-based approach to large horizon rendering is an important step in this direction, since it facilitates the efficient integration with multi-resolution volume ray casting.

Acknowledgments

This work was supported in part by the VRGeo Consortium and the German BMBF InnoProfile project "Intelligentes Lernen". The seismic data set portrayed in this work is courtesy of Landmark/Halliburton.

References

1. Pajarola, R., Gobbetti, E.: Survey on Semi-regular Multiresolution Models for Interactive Terrain Rendering. The Visual Computer **23** (2007) 583–605
2. Dick, C., Schneider, J., Westermann, R.: Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering. Computer Graphics Forum **28** (2009) 67–83
3. Dick, C., Krüger, J., Westermann, R.: GPU Ray-Casting for Scalable Terrain Rendering. In: Proceedings of Eurographics 2009 - Areas Papers, Eurographics (2009) 43–50

4. Musgrave, F.K.: Grid Tracing: Fast Ray Tracing for Height Fields. Technical Report RR-639, Yale University, Department of Computer Science (1988)
5. Cohen, D., Shaked, A.: Photo-Realistic Imaging of Digital Terrains. Computer Graphics Forum **12** (1993) 363–373
6. Cohen-Or, D., Rich, E., Lerner, U., Shenkar, V.: A Real-Time Photo-Realistic Visual Flythrough. IEEE Transactions on Visualization and Computer Graphics **2** (1996) 255–265
7. Qu, H., Qiu, F., Zhang, N., Kaufman, A., Wan, M.: Ray Tracing Height Fields. In: Procedings of Computer Graphics International. (2003) 202–207
8. Oliveira, M.M., Bishop, G., McAllister, D.: Relief Texture Mapping. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH '00), ACM (2000) 359–368
9. Policarpo, F., Oliveira, M.M., Comba, J.L.D.: Real-time Relief Mapping on Arbitrary Polygonal Surfaces. In: Proceedings of the 2005 symposium on Interactive 3D graphics and games. I3D '05, ACM (2005) 155–162
10. Policarpo, F., Oliveira, M.M.: Relief Mapping of Non-Height-Field Surface Details. In: Proceedings of the 2006 symposium on Interactive 3D graphics and games, ACM (2006) 55–62
11. Donnelly, W.: Per-Pixel Displacement Mapping with Distance Functions. In Pharr, M., ed.: GPU Gems 2. Addison-Wesley (2005) 123–136
12. Dummer, J.: Cone Step Mapping: An Iterative Ray-Heightfield Intersection Algorithm. http://www.lonesock.net/files/ConeStepMapping.pdf (2006)
13. Oh, K., Ki, H., Lee, C.H.: Pyramidal Displacement Mapping: a GPU based Artifacts-free Ray Tracing Through an Image Pyramid. In: Proceedings of the ACM symposium on Virtual reality software and technology. VRST '06, ACM (2006) 75–82
14. Tevs, A., Ihrke, I., Seidel, H.P.: Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering. In: Proceedings of the 2008 symposium on Interactive 3D graphics and games, ACM (2008) 183–190
15. LaMar, E., Hamann, B., Joy, K.I.: Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In: Proceedings of IEEE Visualization 1999, IEEE (1999) 355–361
16. Kraus, M., Ertl, T.: Adaptive Texture Maps. In: Proceedings of SIGGRAPH/EG Graphics Hardware Workshop '02, Eurographics (2002) 7–15
17. Lefebvre, S., Darbon, J., Neyret, F.: Unified Texture Management for Arbitrary Meshes. Technical Report RR5210-, INRIA (2004)
18. Lefebvre, S., Hornus, S. Neyret, F.: Octree Textures on the GPU. In: GPU Gems 2, Addison-Wesley (2005) 595–613
19. Hollemeersch, C., Pieters, B., Lambert, P., Van de Walle, R.: Accelerating Virtual Texturing Using CUDA. In Engel, W., ed.: GPU Pro: Advanced Rendering Techniques. A. K. Peters, Ltd. (2010) 623–641
20. Carmona, R., Froehlich, B.: Error-controlled Real-Time Cut Updates for Multi-Resolution Volume Rendering. Computers & Graphics **In Press** (2011)
21. Plate, J., Grundhöfer, A., Schmidt, B., Fröhlich, B.: Occlusion Culling for Sub-Surface Models in Geo-Scientific Applications. In: Joint Eurographics - IEEE TCVG Symposium on Visualization, IEEE (2004) 267–272