International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 3 (2011), pp. 820–827

GPU-Aware Hybrid Terrain Rendering

Christian Dick[1], Jens Krüger[2] and Rüdiger Westermann[1]

[1]Computer Graphics and Visualization Group,
Technische Universität München, Germany
{dick, westermann}@tum.de

[2]Interactive Visualization and Data Analysis Group,
DFKI Saarbrücken, Germany
jens.krueger@dfki.de

Abstract: **We present a hybrid GPU rendering pipeline for high-resolution textured terrain fields that reduces polygon throughput limitations on the GPU. This pipeline uses rasterization and ray-casting in every frame simultaneously to determine eye ray/triangle intersections. It allows shifting workloads flexibly between the triangle setup stage and the parallel execution units, depending on the GPU capacities. We employ a hierarchical tiling of the domain and generate for each tile an error-controlled adaptive tessellation of the terrain. In each rendering frame, for each tile the number of triangles representing this tile as well as the number of pixels covered by the tile are determined, and this information is used to select the fastest rendering method on the executing GPU. Compared to pure rasterization or ray-casting, performance gains of a factor between 2 and 4 are demonstrated for high-resolution fields.**

Keywords: Terrain rendering, graphics hardware, hybrid rendering, ray-casting, rasterization.

I. Introduction and Contribution

Especially in terrain rendering one is confronted with models at a resolution that vastly exceeds the maximum interactive triangle setup rate on current rasterization hardware. In recent work it has been shown that available terrain fields can require more than 30 million triangles to be rendered per frame to achieve a screen-space error below one pixel on high-resolution display systems [3]. Considering the evolution of GPU performance over the last years, fill rate and operational throughput have been substantially increasing, but the growth of polygon throughput has been rather moderate due to the serial implementation of triangle setup in the rasterization stage. For high-resolution terrains, many triangles have to be rendered that are only one pixel in size or less. Triangle setup thus cannot be amortized over many fragments, leading to a severe performance bottleneck.

For such models it was demonstrated that GPU-based ray-casting can speed up the rendering process significantly. The regular grid structure underlying terrain fields enables implementing efficient ray traversal and intersection algorithms, and since ray-casting is realized in the fragment shader, it can effectively exploit the massive parallelism available on recent GPUs. For lower resolution terrains, however, triangle setup is amortized over many fragments, such that rasterization—which immediately yields eye ray/triangle intersections—is faster.

Since typical terrain fields consist of regions exhibiting high-resolution geometric details and others showing rather coarse geometric structures, in this work we introduce a hybrid GPU pipeline for terrain rendering that selects the most efficient rendering method for each region. The pipeline uses rasterization and ray-casting simultaneously in every frame to generate eye ray/triangle intersections. This is in contrast to previous hybrid rendering approaches, where ray-tracing is solely employed for the simulation of secondary effects like shadows and reflections [2, 8, 12].

Our hybrid GPU rendering pipeline is built upon a tile-based visually continuous terrain rendering approach [4] which hierarchically partitions the terrain field and renders each tile separately at the required resolution. The decision which rendering algorithm to use is made on a per-tile basis, by trading the triangle setup rate against the expected performance of the ray-caster. The respectively fastest method is determined by using an oracle which estimates the rendering time of the tile for each method. Since both rendering methods use the same input data and write to the same output buffers, they can work on the same data structures in an interleaved way.

We show that the hybrid GPU rendering pipeline always performs at least as good as the fastest known rasterization or ray-casting methods. Furthermore, we demonstrate that our approach outperforms these methods for high-resolution terrains by up to a factor of 4. Our experiments are performed on recent GPUs, such as the NVIDIA GT200 and GF100 ("Fermi") architectures, and the ATI RV870 architecture. Notably we show that the hybrid rendering pipeline scales effectively in both the triangle setup rate and the number of processing cores.

II. Related Work

Terrain rendering approaches using rasterization have been studied extensively over the last years. They employ the GPU to render large sets of polygonal primitives, and they differ mainly in the used hierarchical height field representation. There is a vast body of literature related to this field and a comprehensive review is beyond the scope of this paper. However, Pajarola and Gobbetti [9] discuss the basic prin-

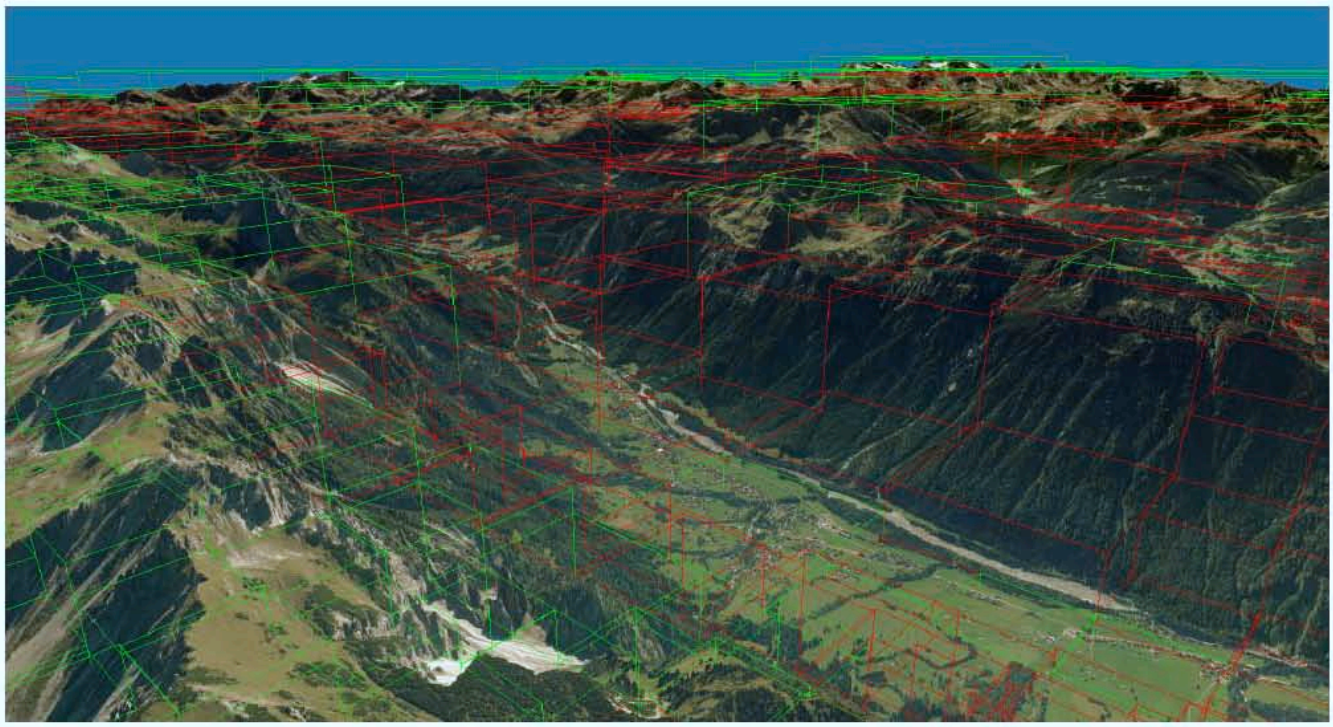


Figure. 1: A 56 km $\times$ 85 km digital terrain field is rendered on a 1920 $\times$ 1080 view port using rasterization (green boxes) and ray-casting (red boxes) simultaneously in every frame. The spatial resolution of the terrain and the texture is 1m and 12.5 cm, respectively. Our hybrid rendering pipeline achieves speed-ups between a factor of 2 and 4 compared to pure rasterization or ray-casting.

ciples underlying such techniques and provide many useful algorithmic and implementation-specific details.

GPU-based ray-casting of height fields given on uniform grids has been reported by Qu et al. [11], Oliveira and Policarpo [7], and Policarpo and Oliveira [10]. Recently, Oh et al. [6], Tevs et al. [13], and Dick et al. [3] performed ray-casting in the regular grid underlying terrain fields using a maximum quadtree on the GPU to speed up the traversal. Ammann et al. [1] proposed the first hybrid height field rendering scheme which uses ray-casting and rasterization simultaneously to enable height field editing.

III. The Hybrid GPU Rendering Pipeline

The hybrid GPU rendering pipeline is built upon the tile-based terrain rendering approach proposed by Dick et al. [4]. This approach employs a tile-based multiresolution representation of the terrain model to achieve visually continuous LOD rendering. Each tile consists of an adaptive triangle mesh and an accompanying photo texture.

In each frame, the tiles inside the view frustum that match a given screen-space error are determined. Our hybrid pipeline then decides for each tile which rendering method—rasterization or ray-casting—is to be used.

For rasterization, the tile's mesh is stored on the GPU as triangle list. Ray-casting a tile is performed as described by Dick et al. [3]. The method operates on a height field representation of the terrain, which is built by rasterizing the tile's mesh into a texture. To efficiently find the ray/height field intersection points, a maximum mipmap acceleration structure is used.

The integration of our hybrid rendering pipeline into a terrain rendering system for very large out-of-core data sets is outlined in Section III-B.

A. *The Rendering Method Oracle*

The hybrid renderer needs to decide whether to use rasterization or ray-casting, depending on which method can render a tile fastest at the current view. Therefore, for each rendering method an estimator function that predicts the rendering time for a particular tile in milliseconds (ms) is used. These estimator functions are parameterized by tile and view dependent properties, and are adjusted to the used GPU by hardware-specific constants. These constants can either be determined directly based on throughput specifications of the architecture and knowledge of the underlying rendering method, or they can be learned in a training phase. In the training phase, the hybrid renderer picks a set of representative terrain tiles and views of the scene, and it renders these tiles using both rasterization and ray-casting. The measured rendering times are then used to determine the GPU-specific constants of the estimator functions.

The estimator functions are designed to be evaluated entirely on the CPU, i.e., we do not use any GPU counters (like the number of fragments passing the depth test) to avoid reading back values from the GPU and thus decreasing the performance due to CPU-GPU synchronization issues.

To predict the time required to render a tile using rasterization we use the following estimator function:

$$\begin{aligned} t_{\text{Rasterize}}(T,F) &\approx \mathcal{O}(T) + \mathcal{O}(F) \\ &\approx c_1 \cdot T + c_2 \cdot F + c_3. \end{aligned} \tag{1}$$

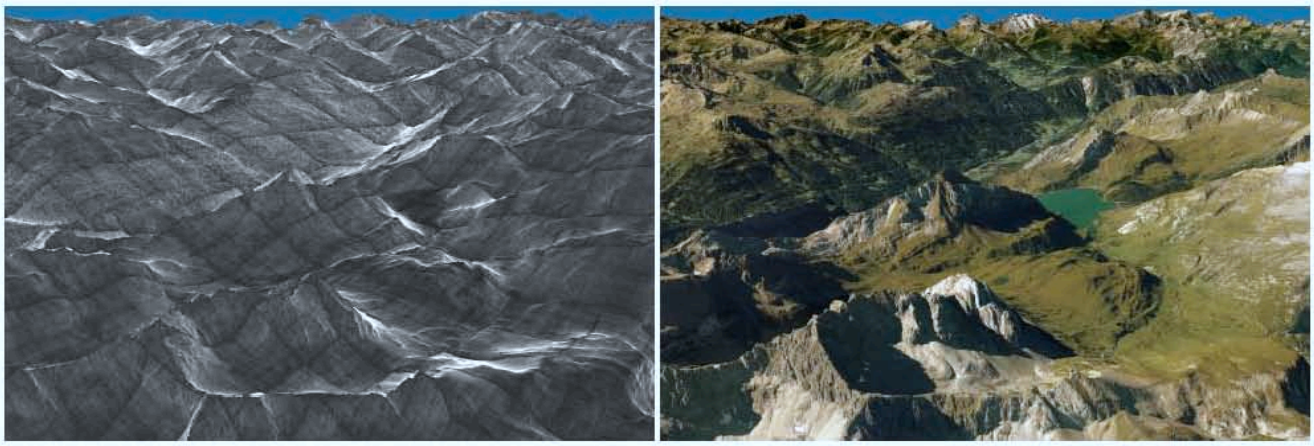


Figure. 2: Ray-casting step counts until ray/surface intersection (left) and the corresponding view (right). The step counts are color-coded from black (0 steps) to white (50 steps).

Here, T denotes the tile's number of triangles and F the number of generated fragments. c_1, c_2 and c_3 are GPU-specific constants (in ms). Since c_2 depends on the complexity of the fragment program—which is negligible in terrain rendering—we set this parameter to zero, reducing the estimator function to

$$t_{\text{Rasterize}}(T) \approx c_1 \cdot T + c_3. \tag{2}$$

The performance of terrain ray-casting is estimated by

$$\begin{aligned} t_{\text{RayCast}}(P, S) &\approx \mathcal{O}(P \cdot S) \\ &\approx c_4 \cdot P \cdot S + c_5. \end{aligned} \tag{3}$$

P denotes the number of rays that are spawned to ray-cast the tile. S is the tile's average number of traversal steps of the maximum mipmap pyramid per ray. Again, c_4 and c_5 are GPU-specific constants. To estimate P the back faces of the tile's bounding box are clipped at the view frustum and the screen-space coverage of the resulting polygons under the current projection is computed. To consider occlusions between tiles, which reduce the number of rays to be spawned, we multiply P by a visibility factor that is obtained from a horizon occlusion algorithm [5]. This algorithm uses a binary tree to maintain a conservative approximation of the horizon based on the tiles' bounding boxes during front-to-back rendering. The visibility factor of a tile is the fraction of the tile's bounding box lying above the current horizon.
Due to the maximum mipmap acceleration structure used for ray traversal, the number of traversal steps of a ray is virtually independent of the distance between the ray's entry point into the tile's bounding box and the ray's intersection point with the terrain surface. Based on a number of experiments we have made on real data sets, we set S to be constant for all tiles, reducing the estimator function to

$$t_{\text{RayCast}}(P) \approx c_4' \cdot P + c_5. \tag{4}$$

This approximation is confirmed by Figure 2 (left), which shows the number of ray-casting steps per pixel for the view shown in Figure 2 (right). As can be seen, the average intensity per tile is roughly constant.

The constants used in the rendering time estimator functions are determined experimentally at the startup of the application by measuring the tiles' rasterization and ray-casting times for a number of randomly selected view positions and directions. The measurements for an ATI Radeon HD 5870 graphics card are presented in Figure 3, which shows the rasterization time vs. the number of triangles (top), and the ray-casting time vs. the (estimated) number of covered pixels (bottom). To obtain the constants, a line is fitted through the measurements, and the constants are derived from the line parameters. Figure 4 shows the resulting rendering time estimator functions for three different graphics cards.
Since the rendering method used to render a specific tile can vary during movement over the terrain, both the triangle list representation as well as the height field representation of the tile's terrain surface may alternately be required. One approach to cope with this issue is to always keep both the tile's triangle list and the corresponding height field in GPU memory. While this reduces CPU-GPU traffic, it drastically increases GPU memory consumption. In our implementation we therefore pursue a different strategy, in that we keep only the currently used representation in GPU memory and re-create the respectively other representation whenever the rendering method is switched. In this way, CPU-GPU bus transfer is slightly increased (see Figure 5), but GPU memory consumption is considerably reduced.

B. System Integration

We have integrated the proposed hybrid terrain rendering pipeline into the out-of-core terrain rendering system described by Dick et al. [4]. This system uses a tile-based multiresolution representation of the terrain model that is built in a preprocess from a height field and a photo texture. First, an average pyramid of the height field and the photo texture is created. This pyramid consists of a hierarchy of successively coarser levels which are constructed from fine to coarse by averaging blocks of 2×2 samples into a sample on the next coarser level, i.e., with each coarser level, the number of samples in each dimension is reduced by a factor of 2. Second, a tile quadtree is built by dividing each level into equally sized,

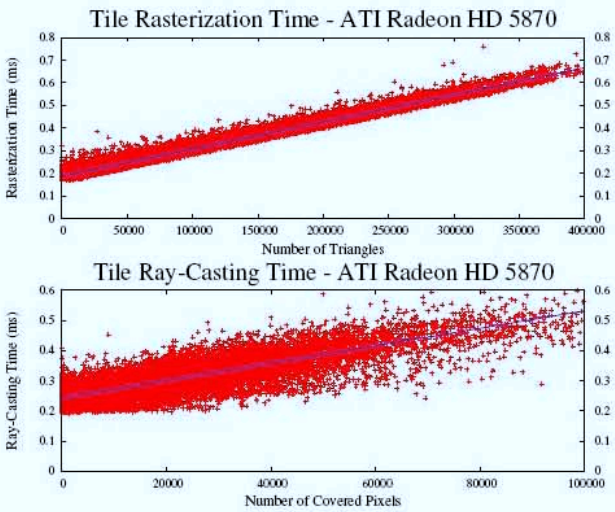


Figure. 3: Top: Measurement of the rasterization time vs. the number of triangles for 75000 tiles. Bottom: Measurement of the ray-casting time vs. the (estimated) number of covered pixels for 75000 tiles.

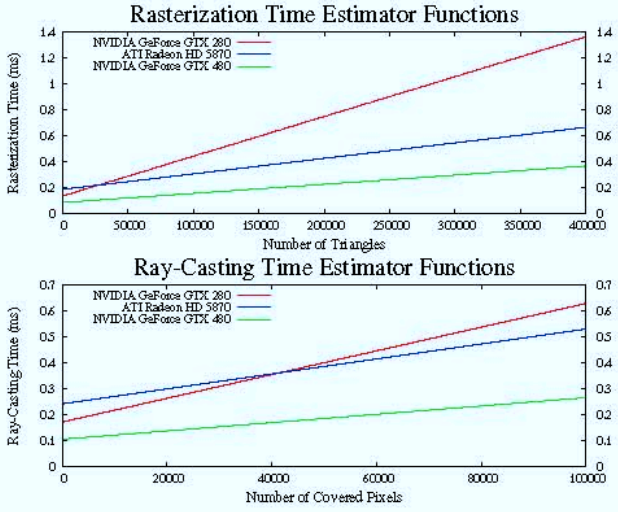


Figure. 4: Rendering time estimator functions for three different graphics cards.

square tiles (using a tile size of 512×512 samples) such that each tile covers 2×2 tiles on the next finer level. For each tile, a restricted quadtree triangle mesh is generated, which approximates the tile's height field within a world-space error tolerance coinciding with the level's grid spacing, i.e., with each coarser level, the world-space error tolerance is increased by a factor of 2. The tiles' meshes and photo textures are stored on disk.

To reduce memory and bandwidth requirements, data compression schemes that can be decoded directly on the GPU are employed. For the restricted quadtree triangle meshes, a compression rate of 8-9 is achieved by using a generalized triangle strip representation to incrementally encode vertex positions. The photo textures are stored using S3TC compression, which offers a fixed compression rate of 6 compared to RGB textures with 24 bits per texel. S3TC is currently the method of choice for terrain photo textures as the GPU supports rendering directly from the compressed representation, including anisotropic filtering.

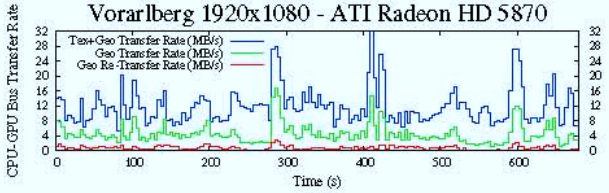


Figure. 5: CPU-GPU bus transfer over the course of a flight over the Vorarlberg data set. Switching between rendering methods requires to re-upload a tile's (compressed) geometry data, which causes additional bus traffic (red). Considering the total bus traffic (blue), this overhead is negligible.

During runtime, the tile tree is partially held in main memory, dependent on the movements of the viewer. A separate IO thread dynamically loads the tiles that are located inside a sphere-shaped prefetching region around the viewer from disk into main memory. Tiles that have left the prefetching region are removed from the tile tree and inserted into an LRU tile cache.

In each frame, the part of the tile tree which is currently available in main memory is traversed in preorder, and the tiles representing the terrain at the current view with a screen-space error of at most 0.7 pixels are determined using view frustum culling and level of detail computation. Our hybrid GPU rendering pipeline then determines for each tile which rendering method is to be used. We then upload all required data into GPU memory that is not already available on the GPU from the previous frames. For each newly visible tile, we upload a mipmap pyramid of its photo texture, assembled from the ancestor tiles in the tile quadtree. In addition, we upload for each newly visible tile as well as for each tile for which the rendering method has changed its compressed mesh. The compressed mesh is instantaneously decoded and stored as a triangle list using 3×32 bits per triangle, or rasterized into a 16 bit UNORM texture to obtain a height field, dependent on whether the tile is to be rendered via rasterization or ray-casting. The compressed mesh is then deleted from GPU memory. Textures and uncompressed geometry data of tiles that are not visible any more are not removed from GPU memory immediately, but are inserted into an LRU cache. The tiles then are rendered in front-to-back order using either rasterization or ray-casting, as determined by our hybrid pipeline.

IV. Results and Discussion

In this section, we give a detailed analysis of the performance of the proposed hybrid rendering pipeline in comparison to pure rasterization or ray-casting. All benchmarks were run on a standard desktop PC, equipped with an Intel Xeon E5506 2.13 GHz processor and 8 GB of RAM. For our tests we used three different graphics cards, an NVIDIA GeForce GTX 280, an ATI Radeon HD 5870, and an NVIDIA GeForce GTX 480 graphics card. For all tests, the far plane was set to 600 km, and the screen-space error tolerance was set to 0.7 pixels.

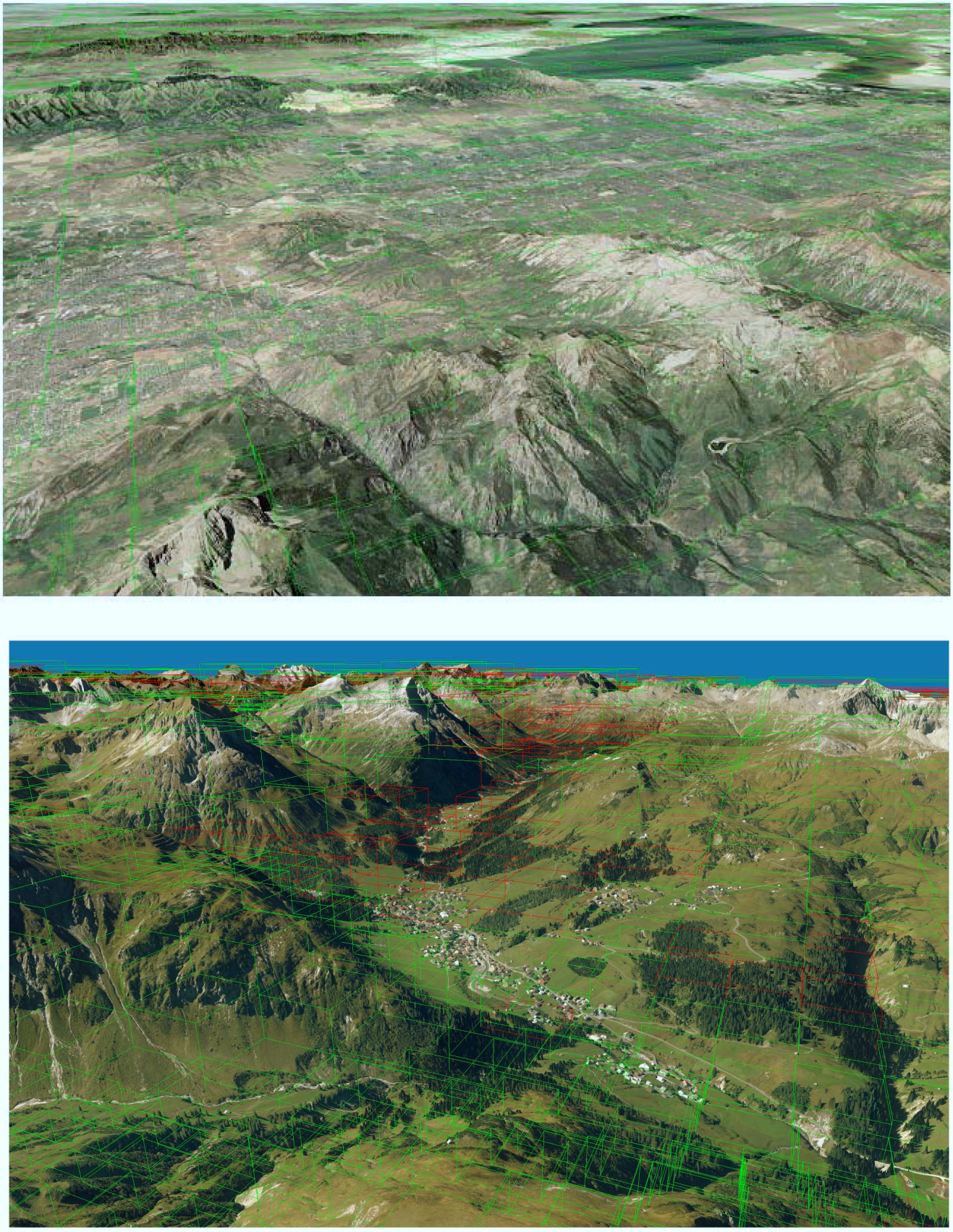


Figure. 6: Screenshots of the Utah (top) and the Vorarlberg (bottom) data set, rendered on a 2560 × 1600 view port. The tile bounding boxes indicate the used rendering method (green: rasterization, red: ray-casting).

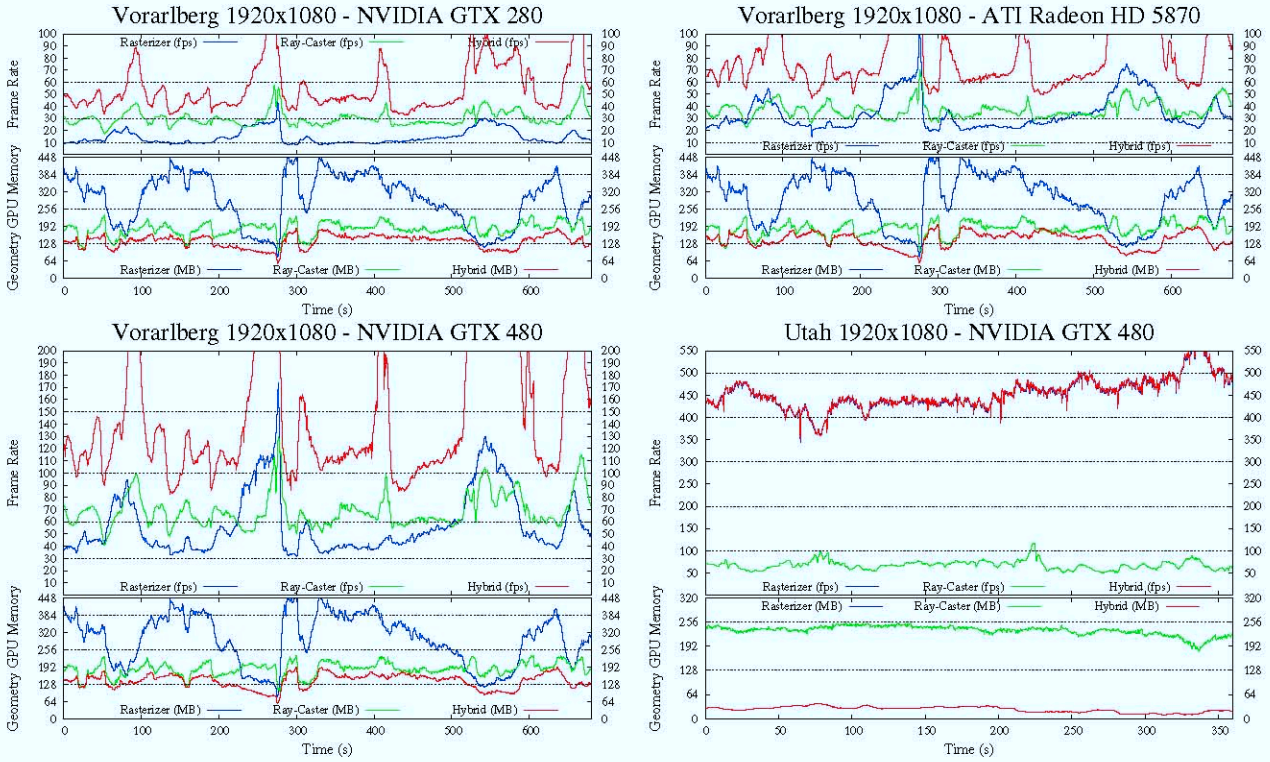


Figure. 7: Comparison of the speed (in frames per second) and GPU memory requirements for geometry (in megabytes) of the hybrid pipeline (red) in comparison to the ray-caster (green) and the rasterizer (blue) over the course of a flight over the Vorarlberg and the Utah data sets using different graphics cards. The view port size was 1920×1080 pixels.

A. *Data Sets*

For our tests, we used two different data sets. The first data set is a digital elevation model of the State of Utah at a resolution of 5 m, accompanied by an orthographic photo texture of 1m resolution (see Figure 6 (top)). With a spatial extent of 460 km $\times$ 600 km, this data set has a size of 790 GB. Contrary to the second data set, its height field does not contain vegetation and buildings, which have been removed by the provider during data processing.

The second data set is a digital model of Vorarlberg, Austria, consisting of a digital surface model at a resolution of 1m and an orthographic photo texture at a resolution of 12.5 cm for a region of 56 km $\times$ 85 km, resulting in a total of 860 GB of data (see Figure 1, Figure 2 (right) and Figure 6 (bottom)). Compared to the Utah data set, the height field of this data set is extremely detailed and clearly exhibits vegetation and buildings.

B. *Performance Analysis and Discussion*

In the following, we first demonstrate the performance of the hybrid rendering pipeline before we present a detailed analysis.

As can be seen in Figure 7, which compares the performance of hybrid terrain rendering to the performance of pure rasterization or ray-casting for different data sets and graphics cards, the hybrid approach is never slower than either alternative, and it usually outperforms both alternatives significantly. While in some situations the speed-up is only marginal, i.e., for low resolution terrain like the Utah data set which is almost solely rendered via rasterization (see Figure 7 (bottom, right)), our new method works extremely well for typical terrain fields exhibiting both low and high resolution regions like the Vorarlberg data set. In these situations the hybrid renderer can speed up the performance about a factor of up to 4.

Figure 7 also shows the GPU memory requirements for geometry (triangle lists and/or height fields) of the three rendering approaches. The hybrid renderer almost always requires the least amount of memory of all alternatives. This looks surprising at first, as we never optimized the hybrid pipeline for memory efficiency, but from the way the decider is designed it becomes clear that the memory requirements are minimized automatically. Since those tiles that are represented by many triangles are ray-cast, these tiles are stored in GPU memory as regular 2D scalar fields and the corresponding triangle data is deleted. Thus, at run-time the most space-efficient representation is automatically selected.

With respect to scalability, Figure 8 shows that with increasing display resolution the hybrid pipeline scales significantly better than pure rasterization or ray-casting. Note that even though the highest display resolution is most likely beyond most of today's typical display capabilities, it plays an important role for supersampling implementations to avoid aliasing on lower resolution screens.

In general it can be said that the performance gain to any method is maximized if the terrain resolution compared to the screen resolution is inhomogeneous, i.e., if some tiles are rather flat—thus containing relatively few triangles—while other tiles exhibit geometric details in the size of a pixel or below.

To verify the prediction accuracy of the rendering method

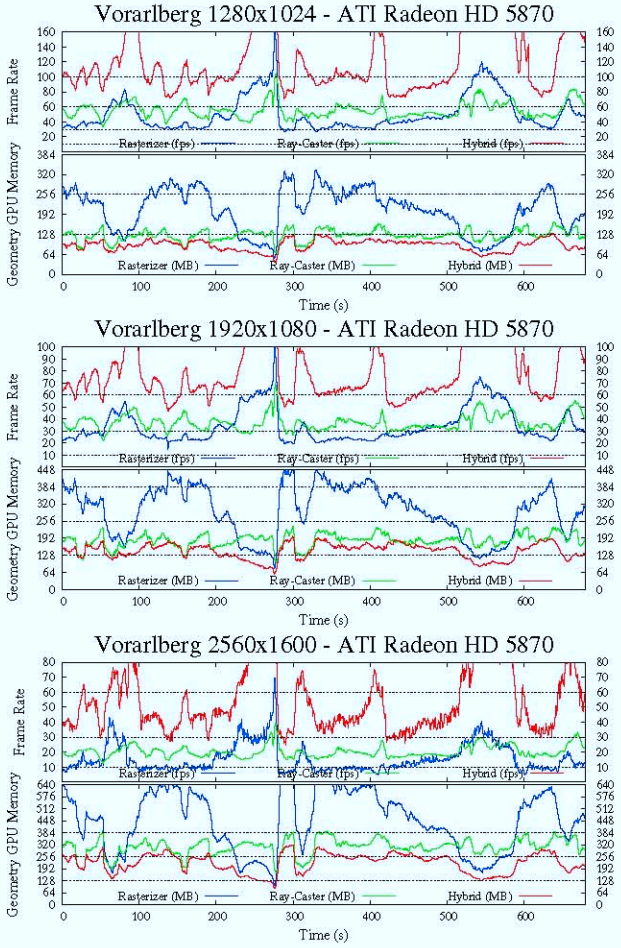


Figure. 8: Rendering performance and memory consumption of the hybrid pipeline (red) in comparison to the ray-caster (green) and the rasterizer (blue) over the course of a flight over the Vorarlberg data set at different screen resolutions. As can be seen the hybrid pipeline not only always performs best, but its advantage grows with higher resolutions. Note that the graphs are scaled differently.

oracle, we experimentally determined the fastest rendering method for 1 million tiles using randomly selected views, and compared the results to the oracle's predictions. For more than 93% of the tiles the oracle correctly predicted the fastest method. Moreover, we observed that by using a perfect oracle, frame rates are increased by only further 2%-5%, showing that the few wrong predictions are insignificant to the overall performance of our hybrid rendering method. The reason is that a wrong prediction typically only occurs when the rasterization and ray-casting times of a tile are almost equal, which means that for those tiles rendering performance is virtually independent of the rendering method.

V. Conclusion and Future Work

In this paper, we have presented a hybrid GPU acceleration pipeline for the interactive rendering of high-resolution terrain fields. We have demonstrated the effectiveness of our approach by comparing it to previous terrain rendering approaches that were based solely on rasterization or ray-casting, respectively. Our comparison showed that for all terrains our hybrid approach not only always performs at least as good as the previously published systems, but often also outperforms them by up to a factor of 4.

To further improve the scalability of the proposed terrain rendering pipeline we will investigate the possibility to use differently sized tiles in this approach. In this way we can adapt more flexibly to the rendering load that is imposed by regions with different height characteristics. In particular, such a tiling has to be based on a precise analysis of the shape of the terrain, resulting in clusters with almost homogeneous resolution.

In the future we will also investigate the applicability of our method to the rendering of arbitrary polygonal models. As our rendering method oracle does not depend on terrain data, we are convinced that it should be applicable to any scene that can be subdivided into separate entities; the only change in the pipeline would be an object-specific ray-casting strategy.

Acknowledgments

The authors wish to thank the Landesvermessungsamt Feldkirch, Austria and the State of Utah for providing high-resolution geo data. The work presented in this paper has been co-financed by the Intel Visual Computing Institute. The content is under sole responsibility of the authors.

References

[1] L. Ammann, O. Génevaux, J.-M. Dischler: Hybrid Rendering of Dynamic Heightfields using Ray-Casting and Mesh Rasterization. In Proceedings of Graphics Interface, pp. 161-168, 2010.

[2] K. Bürger, S. Hertel, J. Krüger, R. Westermann: GPU Rendering of Secondary Effects. In Proceedings of Vision, Modeling, and Visualization Conference, pp. 51-60, 2007.

[3] C. Dick, J. Krüger, R. Westermann: GPU Ray-Casting for Scalable Terrain Rendering. In Proceedings of Eurographics - Areas Papers, pp. 43-50, 2009.

[4] C. Dick, J. Schneider, R. Westermann. Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering, Computer Graphics Forum, 28 (1), pp. 67-83, 2009.

[5] L. Downs, T. Möller, C. H. Séquin: Occlusion Horizons for Driving through Urban Scenery. In Proceedings of ACM Symposium on Interactive 3D Graphics, pp. 121-125, 2001.

[6] K. Oh, H. Ki, C.-H. Lee: Pyramidal Displacement Mapping: A GPU based Artifacts-Free Ray Tracing through an Image Pyramid. In Proceedings of ACM Symposium on Virtual Reality Software and Technology, pp. 75-82, 2006.

[7] M. M. Oliveira, F. Policarpo. An Efficient Representation for Surface Details: Technical Report RP-351, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil, 2005.

[8] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware, Computer Graphics Forum, 26 (1), pp. 80-113, 2007.

[9] R. Pajarola, E. Gobbetti. Survey on Semi-Regular Multiresolution Models for Interactive Terrain Rendering, The Visual Computer, 23 (8), pp. 583-605, 2007.

[10] F. Policarpo, M. M. Oliveira: Relief Mapping of Non-Height-Field Surface Details. In Proceedings of ACM Symposium on Interactive 3D Graphics and Games, pp. 55-62, 2006.

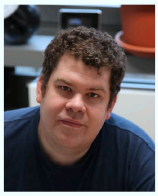
[11] H. Qu, F. Qiu, N. Zhang, A. Kaufman, M. Wan: Ray Tracing Height Fields. In Proceedings of Computer Graphics International, pp. 202-207, 2003.

[12] L. Szirmay-Kalos, B. Aszódi, I. Lazányi, M. Premecz. Approximate Ray-Tracing on the GPU with Distance Impostors, Computer Graphics Forum, 24 (3), pp. 685-704, 2005.

[13] A. Tevs, I. Ihrke, H.-P. Seidel: Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering. In Proceedings of ACM Symposium on Interactive 3D Graphics and Games, pp. 183-190, 2008.

Author Biographies

Christian Dick received the diploma degree in computer science from the Technische Universität München, Germany in July 2007. Since then he is a PhD student in the Computer Graphics and Visualization Group at the Technische Universität München. His current research interests include interactive, physics-based simulation of deformable objects, multigrid methods, GPU computing, medical simulation and visualization, computational steering, as well as the visualization of very large scientific data sets such as terrain data.



Jens Krüger studied computer science at the Rheinisch-Westfälische Technische Hochschule Aachen, Germany where he received his diploma in 2002. The same year, he joined Prof. Rüdiger Westermann's newly established Computer Graphics and Visualization Group at the Technische Universität München, Germany. He finished his PhD in 2006 and after Post Doc positions in Munich and at the Scientific Computing and Imaging (SCI) Institute in Salt Lake City, USA, he joined the Cluster of Excellence on Multimodal Computing and Interaction in 2009 to head the Interactive Visualization and Data Analysis group at the DFKI Saarbrücken, Germany. In addition to his position within the cluster, Jens Krüger also holds an adjunct faculty title of the University of Utah in Salt Lake City and is a principal investigator of multiple projects in the Intel Visual Computing Institute.

Rüdiger Westermann studied computer science at the Technical University Darmstadt, Germany. He pursued his doctoral thesis on multiresolution techniques in volume rendering, and he received a PhD in computer science from the University of Dortmund, Germany. In 1999, he was a visiting professor at the University of Utah in Salt Lake City, and he became an assistant professor at the University of Stuttgart, Germany. In 2000, he was appointed an associate professor at the Technical University Aachen, Germany, where he was head of the Scientific Visualization and Imaging Group. In 2002, he was appointed the chair of Computer Graphics and Visualization at the Technische Universität München. His research interests include real-time physical simulation, general purpose computing on GPUs, interactive data visualization, and real-time rendering.