Scalable Compression and Rendering of Textured Terrain Data

Roland Wahl Manuel Massing Patrick Degener Michael Guthe Reinhard Klein

University of Bonn
Institute of Computer Science II
Römerstraße 164
D-53117 Bonn, Germany

{wahl,massing,degener,guthe,rk}@cs.uni-bonn.de

ABSTRACT

Several sophisticated methods are available for efficient rendering of out-of-core terrain data sets. For huge data sets the use of preprocessed tiles has proven to be more efficient than continuous levels of detail, since in the latter case the screen space error has to be verified for individual triangles. There are some prevailing problems of these approaches: i) the partitioning and simplification of the original data set and ii) the accurate rendering of these data sets. Current approaches still trade the approximation error in image space for increased frame rates.

To overcome these problems we propose a data structure and LOD scheme. These enable the real-time rendering of out-of-core data sets while guaranteeing geometric and texture accuracy of one pixel between original and rendered mesh in image space. To accomplish this, we utilize novel scalable techniques for integrated simplification, compression, and rendering. The combination of these techniques with impostors and occlusion culling yields a truly output sensitive algorithm for terrain data sets. We demonstrate the potential of our approach by presenting results for several terrain data sets with sizes up to 16k x 16k. The results show the unprecedented fidelity of the visualization, which is maintained even during real-time exploration of the data sets.

Keywords

Terrain rendering, level of detail, out-of-core rendering, compression

1. INTRODUCTION

Rendering of textured terrain models has become a widely used technique in the field of GIS applications. Due to the mere size of the data sets, out-of-core techniques must be used to process and visualize such models. Sampling the area of the United States of about 9.2M km^2 with a sampling rate of 10 meters would result in a data set of about 300k x 300k height values. In most cases corresponding texture data is sampled at an even higher resolution. In urban areas sampling rates of 25 cm are common.

To achieve real time rendering without sacrificing accuracy, several aspects have to be considered. On one hand, to exploit the full performance of current GPUs, transmission of large data chunks is advantageous. On the other hand, no unnecessary data should be submitted, since bandwidth and I/O are often the bottleneck of current graphics systems. Furthermore, with the growing GPU power the management of fine-grained LODs on the CPU becomes more and more the limiting factor, and in many rendering applications the GPU is not working at full capacity.

A high-performance terrain rendering system should comprise the following characteristics:

- represent the input data faithfully
- allow for output sensitive rendering, in order to retain scalability (i.e. readily support LODs, occlusion culling, impostors)
- submit and process textures and geometry with adequate granularity to take advantage of GPUs, without taxing the CPU.

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.

- allow for compact storage and on-the-fly decompression of textures and geometry to minimize bus bandwidth and storage requirements.
- local accessibility of geometry and textures without global interdependency, in order to maximize concurrency and to avoid management overhead.

Our method subdivides the geometry as well as the associated textures into equally sized blocks, which we refer to as tiles, and organizes them in a quadtree hierarchy. Tiles from coarser levels correspond to large areas, those from fine levels to small areas. Each geometry tile in the quadtree is represented by a triangulated irregular network (TIN). The vertices are placed on a local regular grid, which has constant resolution for all tiles of the hierarchy. Likewise, textures are stored with constant resolution.

Furthermore, for each tile in our model a guaranteed error bound is available. The approximation error doubles from level to level and is therefore a constant ratio of the tile extent. In contrast to common multi-resolution meshes, which start with a global, coarse approximation of the triangle mesh and decide on a per-triangle basis if further subdivision is necessary, we restrict ourselves to one decision per quadtree cell. This means that each geometry tile only holds one precomputed triangulation, whose connectivity is stored using state of the art compression algorithms. During rendering we perform view-frustum and occlusion culling for the quadtree nodes, which are represented by bounding boxes as long as the geometry is not needed. The accuracy guarantee for the TINs is used to restrict the screen space error to be at most one pixel.

With these techniques we are able to render the simplified data with an image fidelity equal to a rendering of the full-resolution dataset in real time, even if the input data becomes arbitrarily large.

In the next section of this paper, we take a look at related work. In section 3 we give an outline of our algorithm. Section 4 describes how our discrete-LOD model is created in a preprocessing step and section 5 shows how we perform the rendering of our data structure. Then we show some results we deduced from real data sets. Section 7 concludes the paper.

2. RELATED WORK

Fast rendering of terrain datasets with viewpoint adaptive resolution is an active area of research. After the initial approaches by [Gro95, Pup96, Lin96], many different data structures have been proposed. Since giving a complete overview is beyond the scope of this paper, we refer to recent surveys [Lin02, Paj02a] and only discuss the approaches most closely related to our work.

Considering existing approaches for the efficient processing and display of terrain datasets, one can differentiate between two main classes. The first class consists of approaches that employ regular, hierarchical structures to represent the terrain, whereas approaches of the second class are characterized by the use of more general, mainly unconstrained triangulations.

The most established methods of the first class make use of triangle bin-/quadtrees [Lin96, Duc97, Cli01], restricted quadtrees [Paj98, Ger99], RTINs [Eva01] and edge bisections [Lin02]. These structures facilitate compact storage due to their regularity, as topology and geometry information is implicitly defined.

Approaches of the second class use less constrained triangulations. They include data structures like Multi-Triangulations [Pup96], adaptive merge trees [Xia96], hypertriangulations [Cig97] and the adaptation of Progressive Meshes [Hop97] to view-dependent terrain rendering [Hop98]. As proven by Evans [Eva01], TINs are able to reduce the number of necessary triangles by an order of magnitude compared to regular triangulations since they adapt much better to high frequency variations. However, in order to capture irregular refinement or simplification operations and connectivity, a more complex data structure is needed. To alleviate these drawbacks, either Delaunay triangulations [Flo92, Rab97] or a modified quadtree structure have been used to represent irregular point sets [Paj02b].

Since all adaptive mesh generation techniques spend considerable computation time to generate the view-dependent triangulation, the extraction of a mesh with full screen-space accuracy is often not feasible in real-time applications. Many authors have proposed techniques to reduce the popping artifacts due to the insufficient triangle count [Coh96, Hop98] or to amortize the construction cost over multiple frames [Duc97, Hop97, Lin96]. Another approach is to reduce the per-triangle computation cost by assembling pre-computed terrain patches during run-time to shift the bottleneck from the CPU to the GPU like the RUSTiC [Pom00] and CABTT [Lev02] data structures. These methods were further refined, by representing clusters with TINs in a quadtree [Kle01] or bintree domain [Cig03].

To incorporate textures into the above mentioned hierarchies, the LOD management can be either decoupled from the geometry (e.g. the SGI clip-mapping extension and the 3Dlabs Virtual Textures),

which requires special hardware, or they can be handled by explicitly cutting them into tiles and arranging them into a pyramidal data structure [Döl00]. However, this leads to severe limitations on the geometry refinement system, since corresponding geometry has to be clipped to texture tile domains.

3. OVERVIEW

Our method consists of a separate preprocessing stage and the actual rendering stage. A typical input dataset for the preprocessing consists of a digital elevation model (DEM) of the terrain and associated texture maps (e.g. orthophotography).

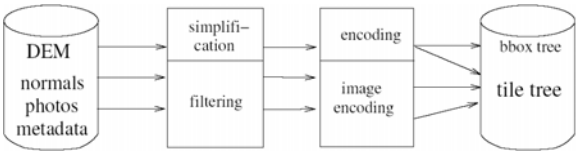

Figure 1. The preprocessing stage.

If desired, a map of surface normals (normal map) can be extracted from the DEM and processed in the same way as the textures. As detailed in the following section, the preprocessing (Fig. 1) recursively builds a LOD hierarchy of tiles (tile tree) through geometry simplification or texture filtering. Finally all resulting tiles are specifically encoded and stored. During geometry encoding, a separate bounding box hierarchy is extracted.

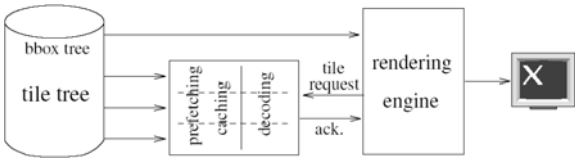

Figure 2. The rendering stage.

Rendering is essentially parallelized among two threads. The main thread selects cells for rendering by considering their visibility and detail. An additional caching thread performs the asynchronous retrieval of associated cell data (e.g. geometry and texture maps). Once all pending requests are completed, the rendering thread hands over the cell data to the graphics hardware. In order to avoid bursts of high workload, the caching thread can also perform prefetching of tiles based on the history of requests or a prediction of the camera path.

Since all operations are handled on a per-tile basis, and no interdependencies among tiles exist, this approach allows for very flexible compression and prefetching schemes.

Therefore, this architecture is able to handle huge terrains, including textures and normal maps. As will be shown in section 5, the number of tiles to be rendered is generally constant. As a consequence, the frame rate is not limited by the amount of input data, but only depends on the complexity of the visible data and on the available graphics hardware.

4. TILE TREE CONSTRUCTION

In this section, we describe how the geometry is processed into a multiresolution data structure, which we call the *tile tree*. Basically, the tile tree imposes a quadtree hierarchy on a set of tiles built from the input geometry and textures. The object space error is bounded throughout the whole pipeline.

The tile tree root holds geometry and texture tiles that cover the whole domain of the dataset, and children partition their parents' domain into equally sized quarters. Texture tiles at the leaves are initialized with the input texture data. Tiles on higher levels are then assembled from their children and downsampled by a factor of 2, that is, the texture resolution remains constant for all tile tree levels. Analogously to the texture sub-sampling process, we partition the input mesh into geometry tiles, which are stored at the tile tree leaves. Geometry tiles on higher levels are built by approximating the input mesh with half the accuracy of their children. We use the symmetric Hausdorff Distances [Kle96] between two meshes as a measure of their approximation accuracy. Both texture and geometry tiles are discretized and compressed before storage.

Error Bounds

All LOD algorithms strive to bound the screen space error, while rendering as few polygons as possible. In the general case, the screen space error ε depends on all viewing parameters: the eye position $\boldsymbol{E}$, the viewing direction $\boldsymbol{n_i}$, the field-of-view ϕ and the screen resolution r.

Since a precise calculation of the screen space error for a tentative simplification is too expensive, one approach is to establish only upper bounds on the object space error δ. The screen-space error can then be easily derived at runtime from the precomputed object space error. From intercept theorems, we have that $\varepsilon = \delta \cdot cos(\alpha) \cdot d_i / d$ where $d_i = cot(\phi) \cdot r / 2$ and $d = (P - E) \cdot n_i$ (fig. 3).

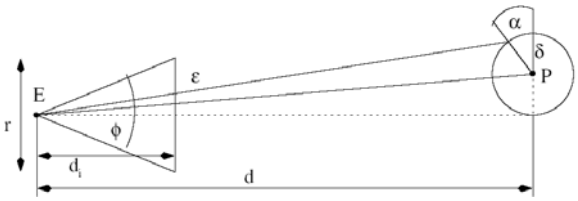

Figure 3. Relationship of errors depicted in 2D.

To further simplify the problem, the direction of the object-space error (i.e. α) is neglected and only its

magnitude δ is regarded. This means that we do not consider the eye position, but only the distance of the observer. We do so for three reasons: First, considering the viewing direction does not save significant amounts of triangles, as Hoppe [Hop98] has pointed out. Secondly, we do not only want to reproduce the correct contours, but also the correct texture coordinates, which requires the object space error to be bounded isotropically anyway[1]. And finally the reduction of dimensions is exactly what we need to build discrete LODs without having too much redundancy in the data.

Consider a tile T with an associated bounding box B. When the object space error for this tile is known to be less than δ_T and we want to guarantee a screen space error below a threshold τ we can use this tile, whenever B lies fully behind a plane with normal $\boldsymbol{n_i}$ and distance $d_i \cdot \delta_T / \tau$.

This means that doubling the observer distance allows us to double the permitted object space error, while maintaining the same screen space error bound. Furthermore, this allows us to represent the geometry of the considered tile on a local grid of constant resolution, because the relative accuracy within a tile is also constant.

In comparison to a continuous view dependent approach (CVLOD), we render a larger number of triangles because the screen space error is overestimated in most places. If one considers an optimal CVLOD mesh, and the mesh complexity falls off quadratically with the permitted Hausdorff error $n \approx \frac{n_0}{\delta^2}$, the number of triangles would remain constant for a fixed viewing direction. In this case, the mesh complexity of our discrete LOD representation would exceed the CVLOD by at most a factor of 4 in the top-down view. When approaching from above, the average overhead would be $\int_1^2 x^2 dx = \frac{7}{3}$, which is at the same time the maximum factor for a lateral view. Since looking from above is the simplest case for rendering (no overdraw, localized texture accesses), the over-estimated mesh complexity does not have a significant impact on performance, and is well worth the cost for the simple, low-cost mesh generation, and the flexibility and complete independence of the data tiles.

Simplification

The geometry simplification starts by splitting the DEM, which typically is given by a regularly sampled heightfield, into equally sized base level tiles (e.g. 129x129 samples each, with overlapping borders). Then, a reasonable triangulation (e.g. regular) is imposed on the height-samples, and a presimplification with error bound δ_{pre} is performed on this mesh. The pre-simplification is meant to accommodate the fact that the input is a regular grid with a given discretization error, so δ_{pre} will be about one half inter-pixel spacing, as this is the amount of uncertainty inherent in the data. These presimplified base-tile meshes are then stored at the leaves of the tile tree, and all subsequent error metrics refer to these meshes.

To make up a tile of the next tile-tree level l, four neighboring tiles are stitched together. The resulting mesh is then simplified to approximate the reference mesh with an error bound δ_l, which is chosen to guarantee an error against the base mesh of $2^l \cdot \tau$. The tile outlines are preserved, but simplifying the borders is allowed if the error implied in the neighboring tile also lies below δ_l. This is an important property, since otherwise the number of border triangles would explode on huge datasets. To avoid unbounded complexity of the reference mesh, which would increase fourfold on every level using a naïve approach, we always measure the Hausdorff error against the penultimate simplification level. That way, the additional error already immanent in the reference can be conservatively estimated as $\frac{1}{4}$, so the overestimation adds up to $1+\frac{1}{4}+\frac{1}{16}+\ldots \leq \frac{4}{3}$. In order to maintain the overall Hausdorff error bound, a conservative estimate of the rounding error committed during compression is subtracted from the permitted simplification error bound for a tile.

The simplification of a tile is highly local, since all measurements during simplification of a tile relate to the tile itself, one of its neighbors, or the corresponding reference tiles.

A parallelization of the simplification is straightforward and the algorithm scales well since the memory requirement for simplifying a tile is bound by a constant. One can even avoid the dependency on neighboring tiles completely if the permitted error along the affected borders is restricted to half the magnitude of the allowed simplification error. That way, the difference between two neighbors is guaranteed to be less than the pixel-threshold, and resulting cracks can be handled as described in section 5.2.

Textures

Bounding the Hausdorff distance between the original and the simplified mesh guarantees the correct representation of contours for a given tolerance, but does not guarantee the correct coloring

[1] Though the L^∞ metric would be sufficient in this case.

of the surface. In addition to conventional decal texture maps, we employ normal maps extracted from the input dataset. With normal maps, shading detail is preserved even in regions of coarse triangulation, which would otherwise be discarded by geometry-based shading (e.g. Gouraud shading). Of course, textures taken from photographs may already contain shaded and shadowed features, but nevertheless normal maps help to reveal the structure of the terrain, especially if additional moving light sources are used.

Since terrain is rather flat, the textures can be projected from above with sufficient accuracy, and the level-of-detail for a texture tile can be chosen in the same way as for geometry tiles. This way, we establish a one-to-one correspondency between texture- and geometry tiles. As already mentioned, texture maps are constructed bottom-up from the input data by downsampling, which basically means building a standard image pyramid on top of the underlying input image (e.g. by averaging 4 neighboring pixels). This also holds for the normal maps, since the defect in length accounts for the roughness of the surface.

During rendering, we apply anisotropic filtering instead of a mip-mapping scheme. This does not only enhance rendering quality, but improves locality because the level of filtering is chosen by the maximum partial derivative.

Compression

One major drawback of using TINs compared to quadtree triangulations [Lin96, Duc97] is that the connectivity is no longer implicit. Fortunately, there are very efficient methods for coding and decoding connectivity [Gum98, Ros99] which rarely use more than 4 bits per vertex. Regarding the coordinates we factor the information into a bounding box – whose xy-coordinates are implicit and whose minimum and maximum elevation are explicitly stored in a separate structure – and a local grid address. As already mentioned before, the grid inside a tile's bounding box may have a constant resolution independent of the level. If we use 129x129-tiles for geometry the inner vertices can be addressed with $14+\lceil \log h \rceil$ bits, where h denotes the height of the bounding box measured in level-dependent units. Typically one will further discretize the bounding box axes with a constant number of bits, so that the rounding procedure does not dominate the Hausdorff error and thereby increase the triangle count. However, all in all the number of bits per vertex even in mountainous terrain rarely exceeds 32 bits per vertex. For experimental results see section 6.

To compress our textures and normal maps, we employ standard compression algorithms such as S3TC and JPEG. S3TC compressed textures offer the great advantage that decoding is implemented on most standard graphics hardware, thus sparing the CPU from decompression. Moreover, they reduce bandwidth and texture memory requirements, as the textures may reside in memory in their compressed form. The main disadvantage of S3TC are block artifacts, which are especially noticeable with normal maps, and the minimal level of control over the compressed image quality. JPEG offers better compression ratios and therefore lessens the load on the I/O, but needs to be decoded in the CPU, which can become a bottleneck. Also, the artifacts are more disturbing. Later standards as JPEG2000 featuring wavelet-codecs are desirable, especially for their inherent support of texture hierarchies, but need to be hardware supported to achieve similar efficiency. Since the tiles can be encoded independently, it is easy to mix different encoding schemes or use lossless formats whenever the signal to noise ratio falls below a certain threshold, but then one needs to take care that the tiles' borders do not become visible due to quality changes.

5. RENDERING

We divide the rendering into two stages, the update stage and the cell rendering stage. During the update stage, the CPU traverses the bounding box hierarchy depth-first and decides which tiles need to be rendered.

Quadtree Update

The update stage can be implemented using a simple top-down traversal of the quadtree hierarchy. Each tile visited in that manner is first checked against the viewing frustum. If the tile's bounding box lies completely outside of the frustum, descent can stop. Otherwise, we need to decide whether the tile in question satisfies our error bound. Since this object-space error bound is fixed throughout a whole quadtree level, and there are no constraints regarding the LOD-difference of neighboring tiles, the selection of an appropriate level of detail is straightforward. All tiles which may be rendered with a LOD of d or coarser lie completely behind a virtual plane which is a shifted copy of the image plane at distance 2^d. If the tile is found to have sufficient detail, it is considered for rendering and, if necessary, geometry, texture and normal map for the tile are requested from the cache. If the tile LOD is not sufficient, we continue our descent.

In a second stage, the tiles found to be visible are rendered.

Repairing Cracks

In case the LOD of two neighboring tiles differ, it is not sufficient to simply render the geometry. Even though the geometric errors between the tiles would fall below the pixel projection threshold, small cracks may become visible due to discretization in the rasterizer stage. But since the cracks are under screen space error control, there is no need to avoid them, they only need to be filled with the correct color. This is achieved by attaching a triangle strip along the border that reaches down the equivalent of one pixel. In this way, the holes are shaded consistently with the borders.

Caching & Prefetching

Even in single-processor system, the CPU, GPU and IO subsystem can work more or less concurrently. To maintain and support such parallelism between the CPU and IO-subsystem, we employ caching and prefetching during the update and rendering stages. During the update stage, the caching thread receives requests from the rendering thread and fulfills them asynchronously. The threads are then synchronized to ensure completion of pending requests. While the terrain is rendered, which is a task independent of the IO subsystem, we can perform node prefetches based on the previously requested nodes and the estimated camera motion. These prefetched nodes are stored in a cache and will in many cases accelerate geometry requests in subsequent update stages.

Output Sensitivity

Since the rendering output always consists of a constant number of colored pixels, achieving output sensitivity is a very demanding task. With our basic LOD algorithm, we achieve, that the per frame complexity is within $O(\log n)$ (i.e. the number of visible tiles per LOD as well as the tile complexity is bounded by a constant). In order to be output sensitive in this theoretic sense, the number of visible tiles has to decrease with growing distance such that its series converges, which basically means that there are only finitely many visible LODs. In fact there are several real world effects, that suggest that this is a feasible demand. Occlusion, earth curvature, atmosphere (fog) and limited flight speed (distant features need not to be redrawn every frame) help to decrease complexity if taken into account. In the following, we will discuss practical aspects of techniques such as occlusion culling and impostors which make use of these effects. The tile granularity combined with the associated object space errors offers advantages for both methods.

Occlusion Culling

During quadtree traversal we can ensure a front-to-back ordering, which enables us to perform per-cell occlusion by conservatively testing tiles against potential occluders.

One can do so by rendering potentially occluding geometry into the depth-buffer (while disregarding texture & color information) during quadtree traversal. Visibility tests on potentially visible cells can be performed by rendering an appropriate enclosure of the geometry and then testing if any pixels passed the depth test. As noticed by Lloyd [Llo02], bounding boxes give satisfying results.

We are able to render the occluders with a greater pixel error than the cells whose visibility is to be determined. This is due to the guaranteed error bounds on our geometry, as the bounding boxes can easily be scaled to compensate for the error introduced by using the coarse occluding geometry. If one accounts for discretization errors it is also possible to reduce the resolution of the depth buffer, thus minimizing fillrate requirements.

Impostors

For a flight speed v there always exists a distance $d(v)$ so that the 3 dimensional effects within a tile or region at this distance are no longer noticeable for several frames. This fact can be exploited by rendering these tiles or regions into textures and project these textures on a quad (impostor) which replaces the geometry. The error is tracked and the impostor is invalidated if the error exceeds a threshold.

If one wants to guarantee a screen space error of one pixel, one has to sum up the errors which are made on the different stages for the impostors. For example if one renders the impostor during setup a certain pixel error is made, but then again during rendering the texture a resampling error is added which also takes account for the resolution of the impostor texture. Both of these errors need to be added to the geometric error which reflects the parallax not represented in the flat geometry.

6. IMPLEMENTATION & RESULTS

For our experiments we implemented a simplifier, a renderer and a coding/decoding module as described in section 3. The simplifier performed edge-collapses, which were generated and scheduled using error quadrics [GH97]. For each proposed collapse the Hausdorff error was computed by calculating the point-triangle as well as the edge-edge distances using the domain as an indicator, which elements need to be checked against each other. Since the z-projection used for finding correspondencies in this

approach does not necessarily yield the closest elements, we establish upper bounds on the error. In order to guarantee linear time-complexity, each collapse is either performed or deleted from the queue.

The rendering was performed on a PC with a 1.8 GHz Pentium 4 processor, 512MB RAM running Linux and a GeForce3 graphics card.

Dataset	Gridsize	Spacing	Time	Filesize	Input
Puget sound	16k x 16k	10m	1h25	9,1MB	256M
Turtmann valley	3 x 4k x 4k	2m	1h12	7,8MB	48M
Westbank	~6k x 15k	10m	0h40	3,8MB	~90M
Grand canyon	2k x 4k	60m	0h02	0,2MB	8M

#Vertices in approximation with relative error					
Dataset	0.5	2	4	8	16
Puget sound	2.698.445	298.271	93.266	47.355	14.921
Turtmann valley	2.014.045	284.462	67.108	26.558	5.969
Westbank	1.135.903	127.713	34.181	14.497	3.266
Grand canyon	65.495	6.868	1.681	654	61

Table 1. Geometry statistics of tested models.

The largest dataset visualized so far with our approach shows the Puget Sound area in Washington, U.S. The input heightmap consists of 16.385x16.385 height samples, with 10m inter-pixel spacing. Additionally, matching texture and normal maps were created. The presimplified dataset, which comprises geometry, S3TC-compressed textures and normal maps, uses 371MB of storage, as opposed to over 1GB needed by the uncompressed heightmap and texture data. Figure 4 depicts framerates of a high-speed (5.400km/h), low-altitude flight over the Puget Sound dataset. Rendering was performed on a 768x576 screen with an error threshold $\tau < 1$ and full resolution normal and texture mapping.

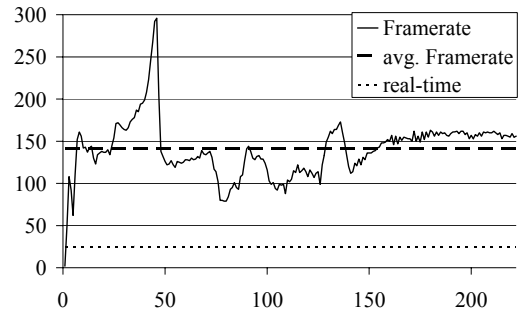

Figure 4. Frame rates for a Puget sound fly-over.

We were also able to visualize a complex dataset of the Turtmann valley in Switzerland (Fig. 5) at high frame rates. The dataset features steep, mountainous parts of the alps at 2 meter resolution. It is actually a digital surface model, which means that even rocks, buildings and trees are present in the geometry. The data was cut into three slightly shifted 4k x 4k datasets and processed into three different tile trees. Note the flexibility of our approach, which easily integrated all three datasets into a single rendering process. We also implemented our terrain rendering engine on a 6-projector powerwall setup, where the isotropic error guarantee extends to accurate depth perception. Videos of the mentioned fly-overs can be downloaded at: http://cg.cs.uni-bonn.de/project-pages/terrain.

Figure 5. Snapshot of Turtmann valley fly-over.

7. CONCLUSION & FUTURE WORK

We have seen that it pays to guarantee conservative Hausdorff error bounds. This enables us to render huge datasets with incredible detail which previous approaches would clearly fail to handle in real-time due to the high triangle complexity. We have shown that off-the-shelf hardware is powerful enough to render huge textured datasets, and are eager to explore the rendering capabilities of our new approach with even larger and more detailed datasets.

8. ACKNOWLEDGEMENTS

We like to thank the Jet Propulsion Laboratory for making their Landsat imagery available on the web for free as well as the Georgia Institute of Technology for the Puget Sound and Grand Canyon datasets. Special thanks to Prof. Dr. Richard Dikau from the Geomorphological and Environmental Research Group who made the Turtmann Valley data available to us.

9. REFERENCES

[Cig97] Cignoni P., Puppo E., and Scopigno R., Representation and visualization of terrain surfaces at variable resolution. The Visual Computer, vol. 13(5), pp. 199-217, 1997.

[Cig03] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchino F., and Scopigno R., BDAM – Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization. Computer Graphics Forum, vol. 22(3), pp. 505-514, 2003.

[Cli01] Cline D., and Egbert P. K., Terrain decimation through quadtree morphing. IEEE Transactions on Visualization and Computer Graphics, vol. 7(1), pp. 62-69, 2001.

[Coh96] Cohen-Or D., and Levanoni Y., Temporal continuity of levels of detail in delaunay triangulated terrain. IEEE Visualization '96, 1996.

[Döl00] Döllner J., Baumann K., and Hinrichs K., Texturing techniques for terrain visualization. IEEE Visualization '00, pp. 227-234, 2000.

[Duc97] Duchaineau M. A., Wolinsky M., Sigeti D. E., Miller M. C., Aldrich C., and Mineev-Weinstein M. B., ROAMing terrain: Real-time optimally adapting meshes. IEEE Visualization '97, pp. 81-88, 1997.

[Eva01] Evans W., Kirkpatrick D., and Townsend G., Right triangulated irregular networks. Algorithmica, vol. 30(2), pp. 264-286, 2001.

[Flo92] De Floriani L., and Puppo E., An on-line algorithm for constrained delauney triangulations. CVGIP: Graphical Models and Image Processing, 54(4), pp. 290-300, 1992.

[Gro95] Gross M., Gatti R., and Staadt O., Fast multiresolution surface meshing. IEEE Visualization '95, pp. 207-234, 1995.

[GH97] Garland M., and Heckbert P. S., Surface simplification using quadric error metrics. SIGGRAPH 97 Conference Proceedings, pp. 209-216, 1997.

[Ger99] Gerstner T., Multiresolution Compression and Visualization of Global Topographic Data. GeoInformatica, 7(1), pp. 7-32, 2003; SFB 256 report 29, Univ. Bonn, 1999.

[Gum98] Gumhold S. and Straßer W., Real time compression of triangle mesh connectivity. SIGGRAPH 98 Conference Proceedings, pp. 133-140, 1998.

[Hop97] Hoppe H., View-dependent refinement of progressive meshes. SIGGRAPH 97 Conference Proceedings, pp. 189-198, 1997.

[Hop98] Hoppe H., Smooth view-dependent level-of-detail control and its application to terrain rendering. IEEE Visualization '98, pp. 35-42, 1998.

[Kle96] Klein R., Liebich G. and and Straßer W., Mesh Reduction with Error Control, Proc. of IEEE Visualization, pp.311-318, 1996.

[Kle01] Klein R. and Schilling A., Efficient Multiresolution Models, in A. Schilling (ed.) Festschrift zum 60. Geburtstag von Wolfgang Straßer, pp. 109-130, 2001.

[Lev02] Levenberg J., Fast view-dependent level-of-detail rendering using cached geometry. IEEE Visualization 2002, pp. 259-266, 2002.

[Lin96] Lindstrom P., Koller D., Ribarsky W., Hughes L. F., Faust N., and Turner G., Real-Time, continuous level of detail rendering of height fields. SIGGRAPH 96 Conference Proceedings, pp. 109-118, 1996.

[Lin02] Lindstrom P., and Pascucci V., Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. IEEE Transaction on Visualization and Computer Graphics, vol. 8(3), pp. 239-254, 2002.

[Llo02] Lloyd B., and Egbert P., Horizon occlusion culling for realtime rendering of hierarchical terrains. IEEE Visualization 2002, pp. 403-409, 2002.

[Paj98] Pajarola R., Large scale terrain visualization using the restricted quadtree triangulation. Technical Report TR 292, ETH Zürich, 1998.

[Paj02a] Pajarola R., Overview of quadtree based terrain triangulation and visualization. Technical Report UCI-ICS TR 02-01, University of California, Irvine, 2002

[Paj02b] Pajarola, R., Antonijuan M. and Lario R., QuadTIN: Quadtree based Triangulated Irregular Networks. IEEE Visualization 2002, pp. 395-402, 2002.

[Pom00] Pomeranz A. A., Roam using surface triangle clusters (rustic). Master's thesis, University of California at Davis, 2000.

[Pup96] Puppo E., Variable resolution terrain surfaces. Eight Canadian Conference on Computational Geometry, pp. 202-210, 1996.

[Rab97] B. Rabinovich, and Gotsman C., Visualization of large terrains in resource-limited computing environments. IEEE Visualization '97, pp. 95-102, 1997.

[Ros99] Rossignac J., Edgebreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics, vol. 5(1), pp. 47-61, 1999.

[Xia96] Xia J. C., and Varshney A., Dynamic view-dependent simplification for polygonal models. IEEE Visualization '96, pp. 327-334, 1996.