Spherical Terrain Rendering using the hierarchical HEALPix grid

Rolf Westerteiger[1,3], Andreas Gerndt[1], and Bernd Hamann[2]

1 **German Aerospace Center (DLR)**
Braunschweig, Germany
`rolf.westerteiger@dlr.de, andreas.gerndt@dlr.de`

2 **Institute for Data Analysis and Visualization & CS Department**
University of California, Davis, USA
`hamann@cs.ucdavis.edu`

3 **Computer Graphics and HCI Group,**
University of Kaiserslautern, Germany

Abstract

We present an interactive spherical terrain rendering system employing a hierarchical subdivision of the HEALPix coordinate system using quadtrees. Compared to other parameterizations, the scheme avoids singularities and allows for efficient fusion of mixed-resolution digital elevation models and imagery. A Level-of-Detail heuristic is used to guarantee both high performance and visual fidelity. Unified treatment of DEM and imagery data is achieved by performing the HEALPix projection within a GPU shader. The system is applied to the exploration of Mars, using both MOLA (NASA) and HRSC (German Aerospace Center) data sets.

1998 ACM Subject Classification I.3.6 Methodology and Techniques – Graphics data structures and data types

Keywords and phrases terrain rendering

Digital Object Identifier 10.4230/OASIcs.VLUDS.2011.13

1 Introduction

Three-dimensional visualization of terrain is a well-studied problem with a history of algorithmic approaches. However, most of these solutions assume a “flat-earth” model, where topography is mapped to a plane. This is acceptable as long as the viewer is close to the surface but breaks down at distances where planetary curvature becomes relevant. Spherical terrain rendering aims to solve this problem by representing the planetary surface as a spheroid to allow for visualization at any scale.

Naive approaches to spherical terrain rendering parametrize the whole surface using a two-dimensional coordinate system. However, any 2D parametrization of the sphere exhibits so-called coordinate singularities, which lead to visible sampling and/or rendering artifacts. For example, in the canonical geographic coordinates (latitude / longitude), singular points appear at $\pm 90^\circ$ latitude (north and south pole).

To avoid these singularities, a 3D parametrization must be used. A common strategy uses a platonic solid as base geometry which is refined using recursive subdivision and extrusion to the spheroid surface. Each of the faces of the base geometry can be parametrized using 2D coordinates while the index of the face can be interpreted as a third (integer) coordinate. To implement Level-of-Detail (LoD) rendering, multi-resolution data structures are used which assign elevation values to vertices in each subdivision level.

Proceedings of IRTG 1131 – Visualization of Large and Unstructured Data Sets Workshop 2011.
Editors: Christoph Garth, Ariane Middel, Hans Hagen; pp. 13–23
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
IRTG 1131

These strategies have in common that the relation between parametric and euclidean coordinates is only given implicitly by the subdivision scheme. As euclidean coordinates are required for rendering, this implies that either the recursive subdivision has to be recomputed in every frame or that meshes have to be stored in euclidean coordinates in GPU memory. The former solution is computationally expensive while the latter incurs a high storage cost.

In this paper we propose applying the HEALPix [5] coordinate system to represent planetary topography for spherical terrain rendering. HEALPix decomposes the sphere into 12 curvilinear quadrilateral base patches of equal area. These are then uniformly subdivided as necessary to form a sampling grid for representing data on the sphere. Compared to other spherical rendering approaches which use implicit coordinate systems, the HEALPix projection is given in a closed form. This property allows for efficient on-the-fly projection of height fields from parametric to euclidean space within a vertex shader. To enable LoD rendering a multi-resolution database structure based on a forest of quadtrees is used. A subset of these trees is kept in GPU memory and updated as the viewer moves by loading data in the background (data streaming).

In the following, we present other work related to spherical terrain rendering. Then we describe the data structure used and give an efficient transformation algorithm to convert geological data sets to this storage scheme. The actual rendering algorithm for DEMs which is based on a top-down traversal of this data structure is subsequently described, including aspects of LoD selection, frustum culling, triangulation of tiles and background data streaming. This algorithm is then extended to also support the visualization of high-resolution imagery draped on top of the DEM.

The system is applied to the interactive exploration of Mars, using a hybrid of MOLA (NASA) and HRSC (German Aerospace) data sets. While HRSC is of higher resolution than MOLA, it does not yet provide full coverage of Mars (as of 2011). We therefore chose to integrate both data sets into a single database, demonstrating that the storage scheme can efficiently capture both at their native resolution.

In Section 4 we present our results using these data and give performance measurements to substantiate the interactivity claim. In the final section, some areas of further research are identified, focusing on aspects of performance and image quality.

2 Related work

Geometry Clipmaps [7] is a planar terrain rendering approach using rectangular, concentric rings of geometry centered around the viewer which decrease in resolution with increasing distance. The algorithm exploits temporal coherence in viewer movement by minimizing per-frame data structure updates using toroidal addressing.

Spherical Clipmaps [3] extend this scheme to spherical rendering by representing the planet's hemisphere which faces the viewer using a set of circular rings. However, due to low accuracy of the $\tan^{-1}$ function on the GPU, this method can produce visible cracks in the final triangulation. Furthermore, compared to the original scheme, vertices are no longer centered on actual height field samples. Due to the additional interpolation required, the data is essentially low-pass filtered when rendering.

The Planet-Sized Batched Adaptive Meshes (P-BDAM [2]) system also uses a subdivision of the sphere into a set of curved base patches. These are then further subdivided using adaptive triangulations. Compared to regular grids, adaptive triangulations require larger per-vertex storage costs but on the other hand need less vertices to represent smooth topography. To further reduce storage costs, the authors suggest an efficient packing scheme for per-vertex data which is unpacked on the GPU during rendering.

Planetary Scale Composition [6] is an interesting approach in which the composition of raster input data is deferred to the rendering stage. A base icosahedron is interactively subdivided and extruded to the sphere surface using a ping-pong buffer scheme on the GPU. When the desired screen resolution has been achieved, subdivision stops and the resulting smooth sphere geometry is displaced by a set of given raster DEMs which are represented using textures. Using this scheme it becomes possible to add new input data on the fly during a running visualization.

Due to numerical instability, the required texture coordinates cannot be computed in a closed form. Instead they are maintained for each vertex as the subdivision progresses. Coordinates for newly generated vertices are interpolated using the haversine geodesic midpoint method. This method is computationally expensive, especially considering that it has to be evaluated once for every vertex in every frame. In our approach data is resampled to the HEALPix grid in an offline preprocessing step. Rendering, however, also requires generation of spherical coordinates which are computed in a numerically stable fashion using the closed HEALPix formulae.

Crusta [1] is a terrain rendering framework which uses a 30-sided polyhedron as base geometry, whose faces are recursively subdivided using a quadtree. Due to the recursive formulation of this geometric construction there is no closed form projection formula between parametric and Euclidean coordinates. This prohibits performing the projection on the GPU and requires computation and storage of Euclidean vertex coordinates for individual tiles at load time. In contrast, our approach only requires storing a single scalar elevation value for each vertex at runtime.

Google Earth is a popular tool for exploring the surface of Earth using data streamed over the internet, which performs well even over low-bandwidth internet connections and provides a high degree of interactivity. However, the system suffers from degenerate triangulations close to the north and south pole, leading to visible artifacts. See Section 4 for a visual comparison with the triangulations generated by our system.

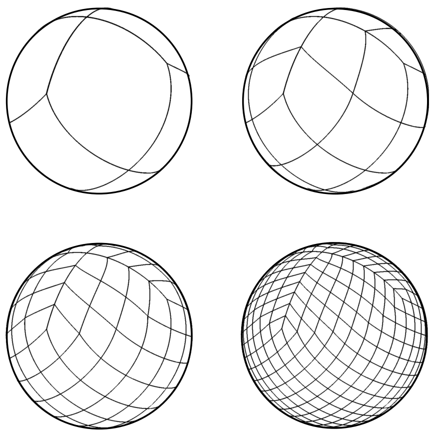


Figure 1 First four levels of the HEALPix sphere tessellation. The root of the hierarchy is formed by 12 curvilinear base patches which are recursively subdivided into quadrants. In our quadtree scheme, each patch corresponds to a 255×255 grid of samples.

The HEALPix [5] scheme is a general solution for representing data on a sphere. It decomposes the sphere into 12 curvilinear quadrilateral patches with associated parametric (u, v) coordinates in $[0, 1]^2$ (see Figure 1). For sampling and data storage, the authors suggest using a hierarchy of uniform (in parametric coordinates) grids for each patch [5]. Samples are stored in quadtree order to optimize referential locality.

Our application demands a high-fidelity model of Mars, which requires composition of multiple data sets with different coverage and resolution. We therefore chose an explicit quadtree data structure to enable representation of sparse and mixed-resolution data.

3 Approach

To represent terrain we use a database consisting of 12 quadtrees, one for each HEALPix base patch. Each tree node stores a tile of 255×255 samples to improve batching in rendering, to reduce management overhead and to increase referential locality during disk I/O while still allowing for 16 bit vertex indices to be used in the representation of the triangle mesh for rasterization.

While the leaf nodes of the tree contain data resampled at the native input resolution, the inner nodes store subsampled representations of their children, forming a multi-resolution data structure suitable for LoD-rendering (see 3.2). Neighboring tiles overlap by one sample at their shared boundary, which incurs a small storage overhead but allows for gapless C_0 continuous rendering without needing to reference neighboring data.

3.1 Data resampling

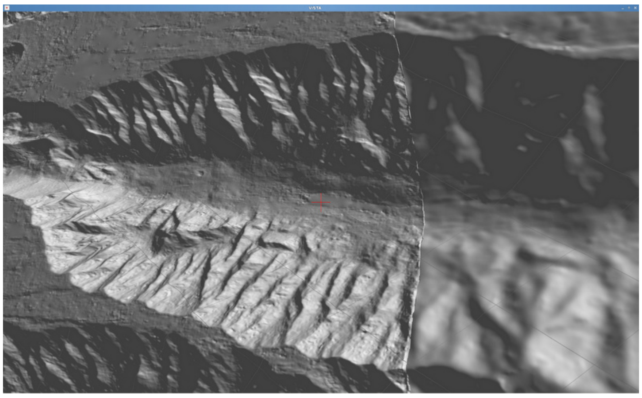


Figure 2 Embedding of high-res into low-res DEM (left: HRSC, right: MOLA).

Digital Elevation Models (DEMs) are frequently represented as georeferenced raster data. These data sets consist of a 2D matrix of height values (the raster data) as well as a so-called georeference which associates samples with their corresponding geographic location.

To transform a given data set to the quadtree database format (described previously), a bottom-up construction process is used. First, an optimal tree depth (resolution) is chosen to faithfully represent the input data. Then the subset of leaf nodes at this level which potentially intersect the data is identified. These nodes are then populated by resampling the raster input and finally the inner nodes are computed by downsampling. This process is repeated for each of the 12 base patches.

The choice of tree depth depends on the resolution of the input data. Due to the equal area property of HEALPix, the grid resolution for a given subdivision level is constant everywhere on the sphere. To faithfully represent the input data, we therefore chose a tree depth d such that the sample density of the leaf nodes is at least as high as in the input data.

In our application individual high-resolution data sets generally only cover a small fraction of the surface. Therefore the set of leaf nodes considered is limited to a subset which is likely to intersect the data. To approximate this set, we construct the boundary curve of the data by projecting each boundary pixel location to geographic coordinates (using the supplied georeference) and then to HEALPix coordinates. We select those leaf nodes which are located inside an axis-aligned box containing all of these points.

For each of the selected leaf nodes, we then iterate over all of the 255×255 sample positions, projecting each sample position first to geographic and then to raster coordinates. Bilinear interpolation within the raster data is used to compute the resampled value. After all intersecting leaf nodes have been populated in this manner, the inner nodes of the tree are derived by iterative downsampling.

Note that raster data sets can designate a special NODATA value, which is assigned to samples having no meaningful measurement. If any of the four input samples used in bilinear interpolation contain this value, the resulting interpolated sample is also marked as NODATA. This value is also assigned if the sampling coordinate (after projection) is not within the bounds of the raster image. If all samples of a node contain NODATA values after resampling, the node is not stored at all.

To support merging multiple data sets, it is also possible to insert data into an existing quadtree database. Already existing nodes are combined with newly generated ones by replacing their sample values. However if an incoming pixel is marked as NODATA, the previous value is kept. This treatment is required because the actual definition domain of many data sets is much smaller than their sampling support, with the difference areas being filled with NODATA samples. Using this scheme, it is possible to insert sparse high-resolution data into an existing low-res DEM database as shown in Figure 2.

The construction process can be trivially parallelized over the 12 base patches, as the quadtrees are mutually independent. By running 12 instances of the construction tool (possibly on different machines) and limiting each instance to only consider tiles within the associated base patch, a database file is generated for each base patch. A separate tool is then used to merge these files into a single database. While this is not optimal in terms of speed up (mainly due to the required I/O for merging the databases), it allows us to generate the hybrid MOLA and HRSC databases within a single day.

The input data sets have a size of 2 GiB for the MOLA DEM, 24 GiB for the HRSC DEM, 54 GiB for red, green and blue channels (HRSC) and 386 GiB for the high resolution B/W nadir channel (HRSC), for a total size of 466 GiB. This data is processed to a set of five databases (DEM, R, G, B, B/W) with a total size of 1.7 TiB.

3.2 Rendering

To render the terrain representation, each of the 12 base-patch quadtrees is recursively traversed in top-down fashion. Recursion stops at a tree node in any of three cases:

1. The node and therefore all of it's children are outside of the viewing frustum. Recursion returns without rendering.
2. The node is sufficiently subdivided to meet the screen space quality requirements (see 3.2.2). The node is rendered and recursion returns.
3. The LoD heuristic decides that further refinement is necessary, however the immediate child nodes are not in memory. In this case, the background I/O thread is instructed to load the four child nodes from disk and the current node is rendered as a placeholder until that data is available.

These cases are tested for in the order given. If none of these occur, recursion continues with the four children of the visited node. This rendering process is stateless in the sense that no information is kept about the set of nodes rendered in the previous frame, minimizing management complexity. In the following, the individual components of this architecture will be described in detail.

3.2.1 Frustum culling

Frustum culling is a technique to eliminate geometry which is located outside of the field-of-view of the camera and therefore guaranteed not to be visible. For performance reasons, bounding geometries are usually employed as proxy objects for this visibility test. Specifically, we follow the classical approach of maintaining an axis-aligned bounding box for each node which is resident in memory. In computing this bounding box, minimum and maximum height values within the node as well as any user-specified height exaggeration factor have to be considered.

Given the geometry of the view frustum, which is a pyramid truncated by two parallel planes, and the extents of the node's bounding box, the separating axis theorem is used to test the two bodies for intersection. This theorem states that, given two convex shapes, an axis exists onto which their projections are separate (non-intersecting) if and only if they are not intersecting.

When testing two polygonal meshes A and B for intersection, the set of axes which need to be tested in this manner is small. Specifically, only the set of face normals and the cross products of all pairs of edges where one edge is taken from A and the other is taken from B need to be considered (see [4]). If the projections of A and B onto any of these axes do not intersect, the original meshes do not intersect either.

The advantage of using axis-aligned bounding boxes is that the set of projection axes is constant for a given view-frustum as the set of face normals and edge directions of the bounding boxes are always the three canonical axes. Therefore, it needs to be computed only once per frame. Furthermore, the bounds of the projection of the view frustum onto this set of axes can likewise be pre-computed, accelerating the test.

3.2.2 LoD heuristic

Level-of-Detail rendering takes advantage of the limited resolution of raster displays by reducing geometric complexity depending on the apparent size of the geometry on the screen. In the visualization of large data sets, LoD techniques are often mandatory to achieve interactive rendering performance. We apply the following conservative approach using the screen-space area of bounding boxes.

One of the goals of the LoD scheme used in this work is guaranteeing maximum visual fidelity. When rendering polygon meshes, this translates into maintaining a geometric resolution of about one vertex per pixel when rasterizing. As our data-structure only allows to select between discrete resolution levels, nodes are refined during rendering until at least the desired level of resolution is reached. A node's bounding box is used to compute a conservative estimate of its screen coverage.

In order to estimate the number of pixels occupied by a node on the screen, the node's bounding box vertices are projected into screen space using the same projection and modelview matrices as used during rendering. Then for each of the six faces of the bounding box the signed area is computed. If the area of a face is negative, it is facing away from the viewer and ignored. All the positive areas are summed up to give the total screen area of the box in

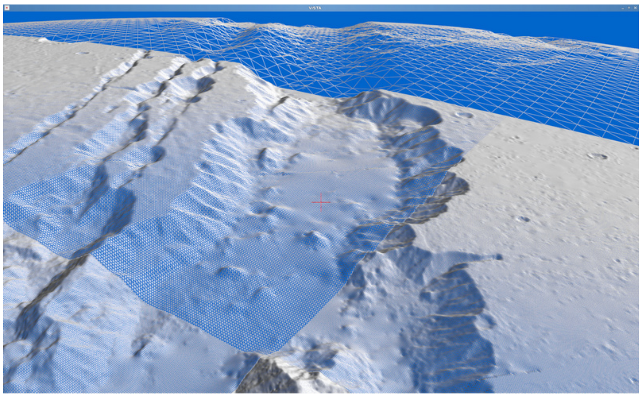


Figure 3 Distance-dependent LoD selection.

pixels, which is always greater or equal to the pixel area of the actual geometry if it were rasterized.

Finally, the estimated pixel area is compared with the number of mesh vertices, which is constant and equal to the number of height samples per node (255×255). If it is smaller, the subdivision level is adequate for the current view and the node is rendered. Otherwise, the recursion continues and the heuristic is applied again to the children.

As a guaranteed resolution of one vertex per pixel is excessive in most use-cases, we provide a user-selectable scaling factor which specifies the desired average number of vertices per pixel. In our experience, a choice of 0.2 provides high visual quality while maintaining good interactive performance.

Figure 3 shows how tree nodes close to the viewer are rendered at a high resolution which decreases with distance. Note that in this example the pixel area threshold was chosen very large for illustration purposes.

3.2.3 Data streaming

In order to guarantee interactivity, it is mandatory to perform slow disk operations asynchronously. For each database we maintain a separate I/O thread which performs these operations in parallel to the render thread. A job queue is used to store read-requests while a result queue contains the loaded data. If during rendering the LoD heuristic decides to refine a node but it's children are not yet in memory, a request is posted onto this queue to load the four child nodes. The I/O thread takes jobs out of this queue and processes them, appending the loaded data to the result queue.

At the beginning of each frame, before starting the actual rendering traversal, the render thread inspects the result queue and inserts any newly loaded nodes into the in-memory quadtree. These nodes are then available for subsequent rendering. This strategy restricts access to the quadtree to the render thread, reducing the complexity of thread synchronization, which is only required for shared access to the job and result queues.

3.2.4 Rasterization

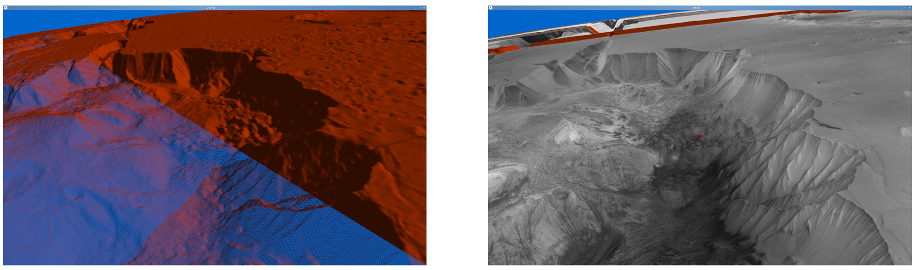
Individual nodes are rendered using triangle meshes interpolating the height field. Each node represents a square sub-region in the parameter space of its associated HEALPix patch. The coordinates of individual mesh vertices are derived by equidistant interpolation within this region. For rendering, mesh vertices are projected to geographic coordinates (latitude / longitude) using the HEALPix formulae. These coordinates are then combined with the corresponding height samples (radii) to form polar coordinates which are eventually converted to Euclidean coordinates. Storing vertex coordinates in Euclidean coordinates for each resident node would be expensive, however. Instead, we perform the coordinate system conversion on the fly during rendering using a vertex shader.

Our approach uses a single 2D proxy mesh which is a uniform tessellation of the $[0,1]^2$ unit square. As this mesh is re-used for rendering each tree node, it's memory footprint is negligible. Per-node elevation data is stored in a 255×255 scalar texture. For each node, the vertex shader implements the following four steps to transform the proxy mesh to the final geometry:

1. Transform mesh to proper sub-region in parameter space
2. Convert parametric coordinates to geographic (latitude, longitude)
3. Read height values out of texture and generate spherical coordinates (r, ϕ, θ)
4. Convert to Euclidean coordinates (x,y,z)

Texture coordinates are centered on the height samples (texels), guaranteeing that the sample points are interpolated by the geometry. To implement shading, normal vectors are estimated using central differencing of the height field. To avoid analytical computation of the u, v direction vectors necessary for the normal estimation, we pre-compute them for the corners of each tile and pass these so-called tangent space matrices to the shader, which interpolates them across the patch.

3.3 Imagery overlay



(a) Shaded DEM (wireframe vs. opaque) **(b)** DEM with high-res BW imagery

Figure 4 Shading vs. Texturing.

In the following we extend the approach previously presented for rendering DEMs to incorporate imagery. Imagery data is processed in the same way as DEMs, producing parallel quadtree databases. The advantage of not combining DEM and imagery into a single database is that both can be arbitrarily mixed and matched at runtime.

In the rendering traversal, DEM and imagery database nodes are now visited in parallel. A straightforward approach to render each pair of data is to extend the vertex shader to assign vertex colors from an imagery texture. However, this is not sufficient due to the fact that imagery data in our applications is of higher resolution than the DEM. These additional levels of resolution are never displayed in this scheme, as the visualization is constrained by the DEM resolution.

Therefore, we chose a different approach which is illustrated in Figure 5. We introduce a parameter Δ_h which specifies the maximum resolution difference between a DEM node and

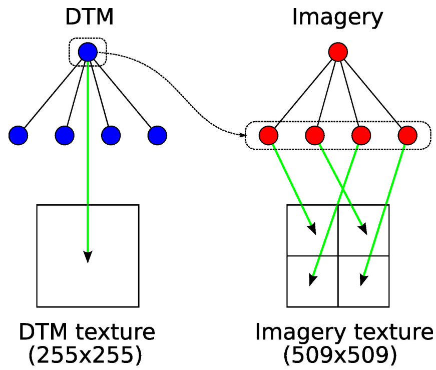


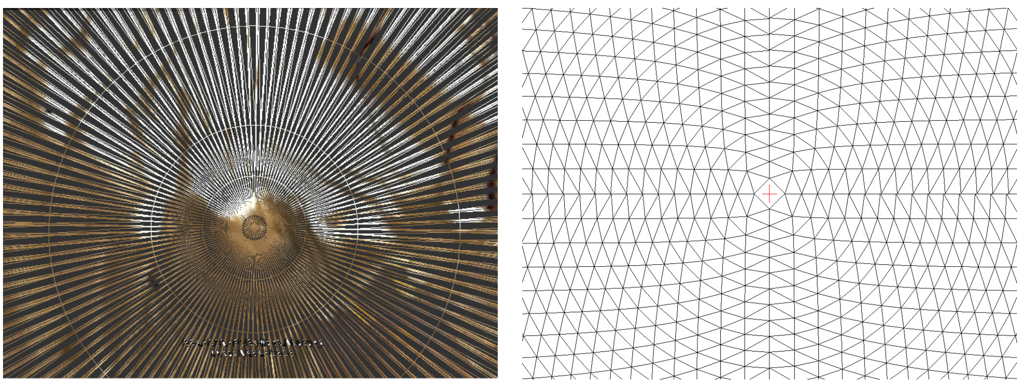
Figure 5 Merging multiple imagery nodes into single texture ($\Delta_h = 1$).

the corresponding imagery. When rendering, the DEM and imagery tree are traversed in parallel as described above. However, instead of displaying the imagery node at the same resolution level as the DEM node, we recurse further into the imagery database and collect all child nodes of degree Δ_h. We then merge the corresponding tiles into a single large texture. Figure 5 shows an example for $\Delta_h = 1$. Regular (fragment stage) texture mapping is then used to provide additional visual detail without increasing geometric primitive count.

The selection of which databases to display can be made at runtime. To simplify switching to another set of channels, we simply delete the resident quadtrees from memory and let the rendering traversal (re-)load any nodes required for the current view, which usually takes less than a second.

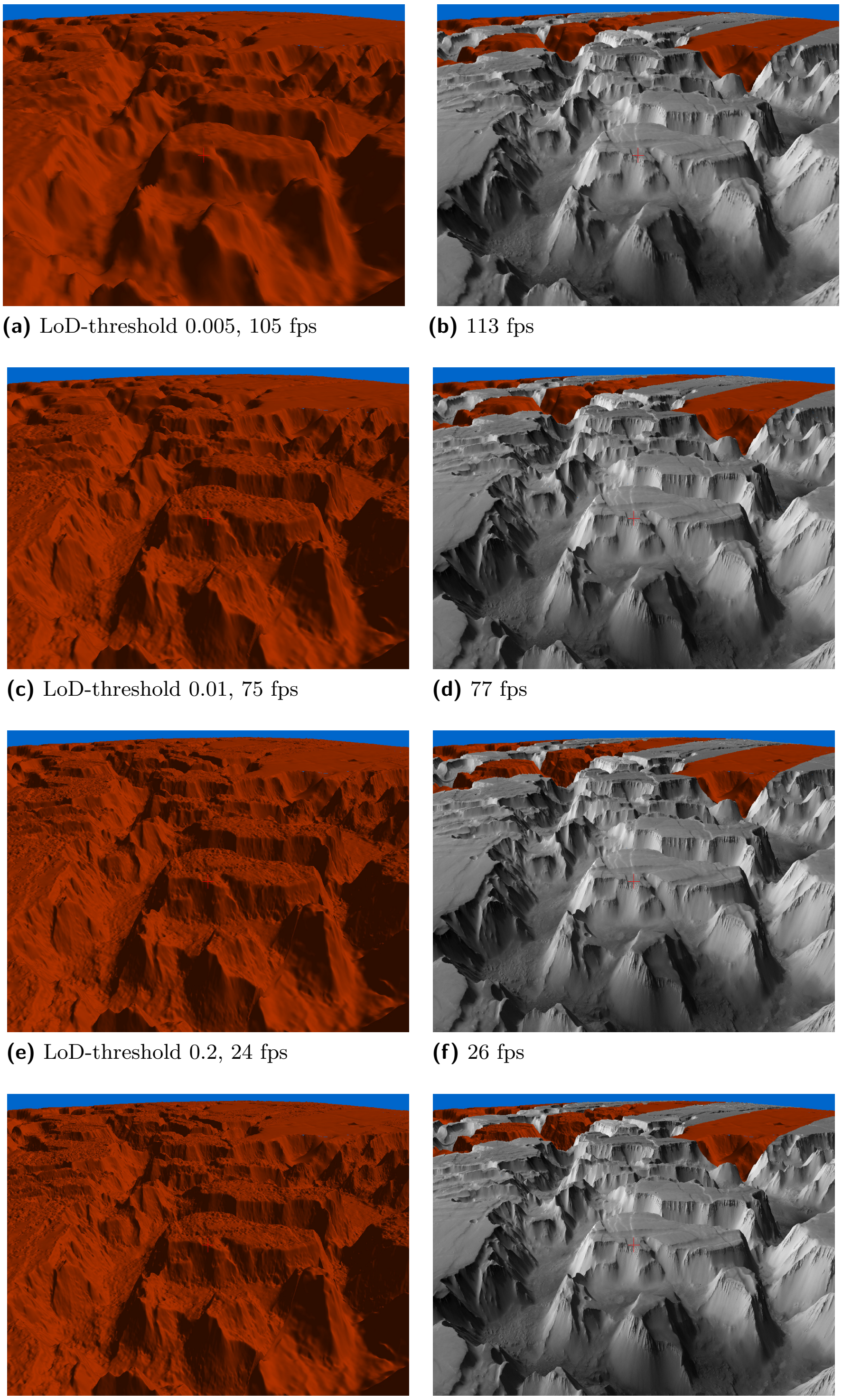
4 Results

Figure 6 shows the triangulation quality in the vicinity of the poles, comparing Google Earth to our approach. Google Earth produces a bad triangulation consisting of long, skinny triangles due to an obvious coordinate singularity at the pole. In contrast, using a uniform sampling of the parametric HEALPix coordinates yields a well-behaved triangulation consisting of equal-area triangles with good aspect ratios.



(a) Google Earth **(b)** Our system

Figure 6 Comparison of triangulations at the north pole.



(a) LoD-threshold 0.005, 105 fps (b) 113 fps

(c) LoD-threshold 0.01, 75 fps (d) 77 fps

(e) LoD-threshold 0.2, 24 fps (f) 26 fps

(g) LoD-threshold 1.0, 9 fps (h) 10 fps

Figure 7 Visual quality and rendering performance at different LoD-thresholds (left: shaded DEM, right: DEM textured with high-res B/W channel ($\Delta_h = 2$)).

Figure 7 shows the surface of Mars, visualized at different LoD-thresholds. Window size is 1024×768 pixels in each case. While rendering performance is not fully interactive at a threshold of 1.0 (min. 1 triangle per pixel), visual quality at 0.2 is not noticeably worse while providing interactive frame rates. The fact that adding imagery textures does not significantly affect the results indicates that performance is limited by GPU geometry processing performance.

While image quality degradation is obvious when comparing the shaded DEMs at thresholds 0.2 and 0.01, it is hardly noticeable when the same geometries are compared with imagery texturing. Therefore, very low LoD-thresholds can be used when imagery is present, resulting in highly interactive frame rates.

5 Conclusion and future work

We have presented an interactive terrain rendering architecture using the HEALPix coordinate system to provide spherical rendering without singularities. By performing critical computations on the GPU, both memory consumption and management complexity are reduced compared to other schemes.

Possible future extensions including incorporation of LiDAR data by using scattered data interpolation schemes to resample data to our uniform grids, locally refining the tree depending on sample spacing.

In the algorithm presented, normal vectors at tile boundaries are discontinuous, resulting in small rendering artifacts. To solve this problem and to provide additional visual detail a low LoD-thresholds, we want to implement normal mapping at a higher resolution than the topography, using the same scheme currently used for imagery overlays.

References

1. Tony Bernardin, Eric Cowgill, Oliver Kreylos, Christopher Bowles, Peter Gold, Bernd Hamann, and Louise Kellogg. Crusta: A new virtual globe for real-time visualization of sub-meter digital topography at planetary scales. *Computers & Geosciences*, 37(1):75–85, 2011. Virtual Globes in Science.
2. Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Planet–sized batched dynamic adaptive meshes (p-bdam). In *Proceedings IEEE Visualization*, pages 147–155, Conference held in Seattle, WA, USA, October 2003. IEEE Computer Society Press.
3. Malte Clasen and Hans-Christian Hege. Terrain rendering using spherical clipmaps. In *EuroVis*, pages 91–98, 2006.
4. S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 171–180, New York, NY, USA, 1996. ACM.
5. K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. Healpix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2):759, 2005.
6. R. Kooima, J. Leigh, A. Johnson, D. Roberts, M. SubbaRao, and T.A. DeFanti. Planetary-scale terrain composition. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):719–733, sept.-oct. 2009.
7. Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 769–776, New York, NY, USA, 2004. ACM.