Hardware Accelerated Terrain Rendering by Adaptive Slicing

Stefan Röttger, Thomas Ertl

Visualization and Interactive Systems Group
University of Stuttgart, Germany *

Abstract

Many terrain rendering algorithms have been developed, which perform a variety of mesh optimizations to achieve interactive frame rates. The most prominent approach is the so called continuous level of detail technique, which approximates a terrain by computing a view-dependent triangulation. One disadvantage of this approach is the fact that the rendered terrain is of course just an approximation of the original data set. By exploiting the capabilities of todays mainstream PC graphics accelerators we propose a new technique for the exact rendering of height fields. Due to its relationship with volume slicing we call it adaptive terrain slicing. This technique is adaptive in the sense that a bundle of slices is used to sample the terrain, whereas the number of slices is determined to assure sub-pixel accuracy. The core of our approach is a hierarchical bounding box representation of the terrain, which is traversed in a top-down order. During traversal we calculate the actual rendering costs of our adaptive terrain slicing approach, which enables us to decide whether it would be faster to render the contents of the actual bounding box or to descend further down the hierarchy. After that the minimized terrain slicing costs are compared to the cost of polygonal rendering and the faster method of both is applied. Since the speed of our terrain slicing approach is limited mainly by the rasterization bandwidth of the graphics hardware, we can efficiently decouple the rendering costs from geometric complexity leading to high frame rates without compromising image quality. In particular, our approach is well suited for replacing the commonly used bump maps with the visually more pleasing displacement maps.

*Universität Stuttgart, Institut für Informatik, Abt. VIS, Breitwiesenstraße 20-22, 70565 Stuttgart, Germany, E-Mail: {Stefan.Roettger | Thomas.Ertl}@informatik.uni-stuttgart.de.

1 Introduction and Motivation

Although terrain rendering is a research area with a long history it is still a topic of actual interest, especially in GIS (Geographic Information Systems) and interactive entertainment. Among the first attempts to visualize a terrain, acquired by satellite imagery or cartographic digitization for example, resulted in the utilization of the so called TINs (Triangulated Irregular Networks [18]). By generating an approximate triangle mesh the number of polygons was reduced to achieve interactive frame rates. The most important disadvantage of such an approach is the usage of an object space approximation criterion that did not take advantage of the special properties of a terrain, that is the huge lateral extent of a terrain in comparison to its height. More advanced visualization algorithms are thus using a screen space based approxiation criterion leading to a view-dependent triangulation. The roots of these algorithms are going back as far as to the mid 70ies. Since then flight simulators have been using levels of detail methods to accelerate terrain rendering. Nowadays, the most prominent technique is the so called continuous level of detail approach [10, 11, 3, 15, 13], but other well known techniques like progressive meshes [5, 6, 7] or wavelets [4] have been applied to terrain rendering as well to exploit the benefits of view-dependent triangulations.

In order to eliminate a visually irritating drawback of these view-dependent algorithms, the so called popping effect, LOD interpolation schemes have been introduced in [1, 15, 7, 13], which are commonly known under the term geomorphing. All these approaches spend a great deal of the rendering time to compute the view-dependent triangulation. Recent image based approaches, however, are utilizing the rapidly increasing rasterization performance of mainstream PC graphics accelerators to decouple the geometric complexity from rendering

time complexity. Without being exhaustive a list of those algorithms includes layered impostors [16, 2], isosurface extraction with 3D textures [19, 14], vegetation rendering by slicing [8] and relief texture mapping [12]. The relief texture mapping approach uses a separable 2D image warp to preprocess a displacement map, which afterwards can be rendered by standard texture mapping hardware. While this approach would be a fast solution to render a terrain given by a height field or a displacement map, it is still not applicable, because prewarping is not yet supported by graphics hardware and has to be accomplished in software.

In this paper we propose a new technique called adaptive terrain slicing, which allows us to display a terrain or a displacement map at high framerates and on standard PC graphics hardware.

Our slicing approach is based on the elevation map white paper from NVIDIA [17], which shows how to draw horizontal slices through a terrain. Because of the restriction to horizontal slices this approach is suited only for rendering height fields from the distance and from above. A viewer that is located on top of the landscape would be able to easily distinguish single slices. In contrast to that approach we show how to draw arbitrarily oriented slices through the terrain, which enables us to view the terrain from arbitrary angles and positions and with or without perspective projections. A similar technique for hardware accelerated displacement mapping of polygonal models has been presented in [9]. However, the authors did not specifically address the special case of terrain rendering, which leaves much more room for improvement. Furthermore, we are introducing an error measurement by which the number of drawn slices can be adjusted to ensure sub-pixel accuracy. Adaptively choosing the density of the slices enables us to reduce the required number of slices for distant parts of the terrain, where the area of the rendered triangles corresponds to less than one pixel. Thus, by rendering a single terrain slice a considerable amount of triangles can be rasterized.

We also present a hybrid technique, which uses a bounding box hierarchy to adaptively balance the load between terrain slicing and normal polygon rendering, which is chosen for parts of the terrain, where a single triangle maps to much more than a single pixel. These parts can not be rasterized efficiently, because the number of required slices would be much too high to ensure sub-pixel accuracy. In these cases we switch back to normal polygon rendering, which is necessary only in the vicinity of the point of view.

Due to the image based nature of our approach the rendering complexity is almost independent of the geometric complexity of the underlying discrete mesh and is limited mainly by the rasterization bandwidth of the graphics hardware. In comparison to continuous level of detail algorithms we achieve high frame rates without compromising quality and without performing time consuming triangulations.

2 Terrain Slicing

The first step to decouple terrain rendering from geometric complexity is to set up an image based method that is able to sample a height field by means of slicing. With respect to this goal, the first approach that might come into mind is the volume slicing analogue. Here a cartesian volume data set is sampled by rendering a set of parallel slices or concentric shells that intersect a 3D texture map, which contains the Hausdorff distance to the object's surface. Then the polygonal object can be rasterized by simply applying an appropriate transfer function and an alpha test that cuts away the transparent parts of the volume. Because of the huge memory requirements of the volumetric terrain representation this method is not well suited for terrain rendering. Furthermore, 3D textures are not yet supported on every graphics platform.

Nevertheless, it turns out that we can achieve our goal without the use of 3D textures by utilizing a special OpenGL extension and encoding the elevation values of a height field (or displacement map) into the alpha channel of a 2D texture map. Suppose we want to render an arbitrarily oriented slice through the terrain, then this can be accomplished in the following way:

1. Encode the terrain elevations into the alpha channel of a 2D texture mip-map by normalizing the height values to the range $[0, 1]$. The smallest height maps to one and the highest elevation maps to zero. The red, green and blue channels can be used to represent a perspectively rectified photo (ortho-image) of the terrain in the usual way.

2. Set up the texture coordinate generation stage of OpenGL, so that a 3-space vertex is projected automatically onto the x-z plane, and the derived texture coordinates s and t are mapped to the correct coordinate range $[0,1] \times [0,1]$.
3. Initialize the texture rasterization stage by using the OpenGL extension `GL_EXT_texture_env_combine` with the following setup, which has the effect of adding the primary alpha component to the texture alpha component scaled both by a factor of $\frac{1}{2}$:

```
GLfloat color[]=
   {1.0f,1.0f,1.0f,0.5f};

glTexEnvi(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_COMBINE_EXT);
glTexEnvi(GL_TEXTURE_ENV,
          GL_COMBINE_RGB_EXT,
          GL_MODULATE);
glTexEnvi(GL_TEXTURE_ENV,
          GL_COMBINE_ALPHA_EXT,
          GL_INTERPOLATE_EXT);
glTexEnvi(GL_TEXTURE_ENV,
          GL_SOURCE2_ALPHA_EXT,
          GL_CONSTANT_EXT);
glTexEnvi(GL_TEXTURE_ENV,
          GL_OPERAND2_ALPHA_EXT,
          GL_SRC_ALPHA);
glTexEnvfv(GL_TEXTURE_ENV,
           GL_TEXTURE_ENV_COLOR,
           color);
```

4. Enable alpha testing with the comparison mode `GL_LEQUAL` and an alpha value of $\frac{1}{2}$.
5. Draw an arbitrary terrain slice by assigning each vertex an alpha value in the range $[0,1]$ that corresponds to each vertex' height (the y-component of each vertex).

Since we used an inverse mapping to encode the height values into the alpha texture component, this setup effectively computes the difference of each fragment's height and the height of the terrain. In our specific setting a height difference of zero corresponds to a rasterized alpha value of $\frac{1}{2}$. By enabling an alpha test that passes below or exactly at this value a fragment that lies exactly on or below the surface of the terrain will pass the alpha test. As a consequence, only the solid part of the terrain will be rasterized (see also Figure 1). We may also use the texture combiner mode `GL_ADD_SIGNED_EXT` as an alternative to the mode `GL_INTERPOLATE_EXT`, but then we would effectively lose one bit of accuracy leaving only 7 instead of 8 available bits for encoding the heights into the alpha channel of the texture.

The application of mip-maps ensures proper texture filtering. It is also advised to use anisotropic texture filtering, where available. The utilized OpenGL texture combiner extension is available on nearly every mainstream PC graphics card including the ATI Rage 128, the ATI Radeon and the NVIDIA TNT2 up to the GeForce3 graphics accelerators. For this reason, the described hardware accelerated terrain slicing approach will operate on a broad range of personal computers and work stations.

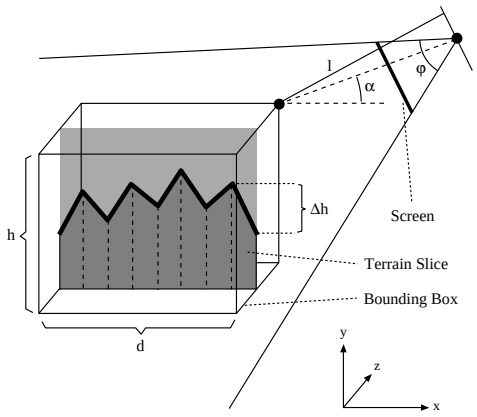

Figure 1: The grey rectangle in the middle of the figure indicates a single vertical terrain slice. Only the solid part of the terrain below the zig-zag line is rasterized. This is achieved by utilizing texture combiners and alpha testing and by encoding the height of the terrain into the alpha channel of a 2D texture map.

3 Adaptive Terrain Slicing with Sub-Pixel Accuracy

In the previous section we described how to draw a single arbitrarily oriented slice through a terrain model. Next we explain how to perform sub-pixel

exact terrain rendering by drawing a bundle of terrain slices. Suppose we want to render only a small part of the height field, which is enclosed by a bounding box. The enclosed part of the terrain can be rasterized by drawing a bundle of parallel slices in the space of the bounding box (see also Figure 1 and 2). Depending on the angle of sight the slice bundle is either aligned to the x-, y- or z-axis, so that the slices are always facing towards the viewer. For a vertical angle of sight higher than $\alpha = 45°$ a horizontal y-slice bundle is drawn, otherwise either a vertical x- or z-slice bundle is computed. In order to improve performance and to take advantage of the early Z-test of modern graphics accelerators, we are sorting the slices in a front to back fashion.

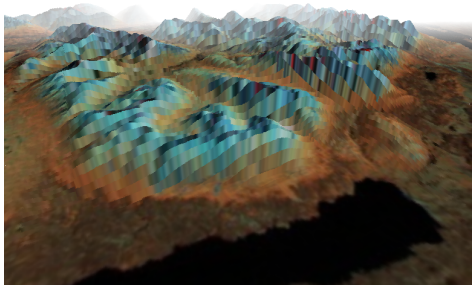

Figure 2: Screenshot of Yukon Territory with reduced accuracy. Therefore, single terrain slices are becoming distinctable.

Now the main question is how to calculate the minimum number of equally spaced slices, such that the rasterization of the terrain will be sub-pixel exact. Clearly this depends on the size of the enclosing bounding box with horizontal size d and vertical size h, the distance to the point of view, the angle of sight, the dimension of the viewport (in pixels) and the field of view as depicted in Figure 1. In the case of a symmetric projection only the height of the viewport p_y and the vertical field of view φ need to be taken into account. Considering this, the projected size (in pixels) of a line segment of length x and average projection distance l, which is seen under an angle of sight α, can be approximated by the following equation with $\theta = \cos\alpha$:

$$x'(x,l,\theta) = \frac{x \cdot \theta \cdot p_y}{l \cdot 2\tan\frac{\varphi}{2}}$$

Using this approximation the minimum required number of slices m_y in a horizontal slice bundle, which is seen under a vertical angle of sight α, can be expressed as follows:

$$m_y = x'(h, \min l, \cos\min\alpha)$$

Here $\min l$ and $\min\alpha$ denote the minimum distance and the minimum vertical angle of sight α of all eight corners of the bounding box. As a result, a consistent lower bound cannot be given for bounding boxes that intersect the near clipping plane. Similarily, the lower bound $m_{x/z}$ for a vertical slice bundle is given by:

$$m_{x/z} = x'(d, \min l, 1)$$

4 Adaptive Terrain Slicing versus Polygon Rendering

Now that we can efficiently rasterize the solid parts of the terrain with sub-pixel accuracy, an analysis of the required number of drawn slices shows that the amount of slices is quite small for distant parts of the terrain, but can grow arbitrarily large for near parts. Here polygon rendering is more efficient. For this reason, we estimate the rendering cost of terrain slicing and polygon rendering and subsequently choose the method, which is faster. In order to fully exploit the advantages of such a hybrid approach, we use a hierarchical bounding box representation of the terrain. For each bounding box we calculate three costs during the traversal of the hierarchy: the cost of brute-force polygon rendering, the cost of terrain slicing, and the sum of the costs for descending to the next hierarchy level. Recursively choosing the action with the lowest associated cost effectively maximizes the overall rendering performance.

Typically, a large bounding box leads to a descendance to the next hierarchy level, because the sum of the rendering costs of the children are much lower than the actual terrain slicing or polgon rendering costs. At a certain hierarchy level descendance does not pay off anymore. Then polygon rendering is chosen for the vicinity of the view point, whereas terrain slicing performes better in the distance (see also Figure 4). In our case the hierarchy is a quadtree, which is constructed from meshes with $2^n + 1$ grid points in each dimension. Other mesh sizes need padding or resampling.

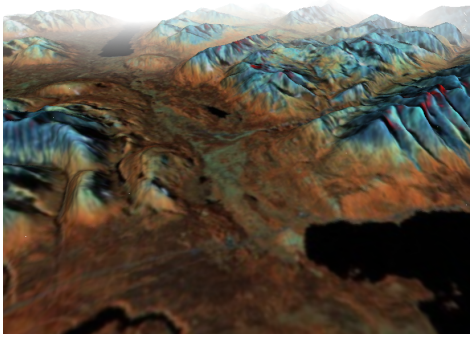

Figure 3: A screenshot of Yukon Territory, Canada, rendered with sub-pixel accuracy by means of our hybrid terrain slicing algorithm.

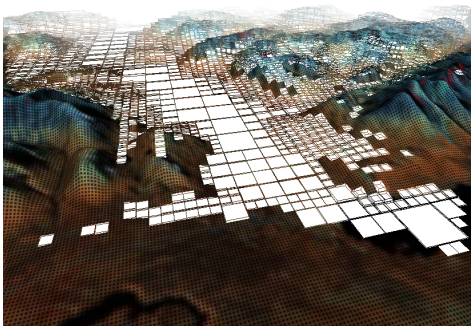

Figure 4: The same screen shot as in Figure 3, but the terrain slices have been replaced with their bounding boxes and the geometry of the triangle fans has been made visible by darkening the center vertex. A larger version of the image can be seen on the color plate.

At first glance, the calculation of the exact rendering costs, and in particular the correct propagation of these values up the quadtree, seems to be a very complicated task. Fortunately, it turns out that we can simplify this task due to the following observation: Descending the quadtree does not increase or lower the total cost of polygon rendering, because the total amount of triangles stays the same. There is only a minor trade off due to the reduced average length of the triangle strips. As a consequence, we can minimize the terrain slicing costs decoupled from the costs of polgyon rendering. This is achieved by means of a simple subdivision test, which compares the terrain slicing cost of the actual node with the sum of the terrain slicing costs of its children times a constant c:

$$\texttt{subdivide}, \texttt{if } \texttt{cost}_{\texttt{node}} > c \cdot \sum_{i=1}^{4} \texttt{cost}_{\texttt{child}_i}$$

At the end of the traversal we need to calculate the polygon rendering costs only once in order to compare against terrain slicing and choose the faster method. The influence of the constant c will become clear at the end of this section.

As we have seen, it basically boils down to accurately estimate the rendering costs of terrain slicing and polygon rendering. Since each graphics accelerator has got its own performance profile, we are measuring two characteristic constants at the start up of the terrain rendering application. These two constants are the number of vertices and the number of pixels per second, which can be processed by a specific graphics accelerator (denoted by `vtx_speed` and `rst_speed`, respectively). With these characteristic constants the time spent by either processing n vertices or by rasterizing a polygonal surface of area A and average distance l, which is seen under an angle of sight α with $\Theta = \cos\alpha$, can be estimated as follows:

$$\texttt{vtx\_cost}(n) = \frac{n}{\texttt{vtx\_speed}}$$

$$\texttt{rst\_cost}(A, l, \Theta) = \frac{A \cdot \Theta}{\texttt{rst\_speed}} \cdot \left(\frac{p_y}{l \cdot 2\tan\frac{\varphi}{2}} \right)^2$$

In the case of using triangle fans instead of triangle strips the polygonal rendering cost of a mesh with size $(s+1) \times (s+1)$ can be approximated by:

$$\begin{aligned} \texttt{cost}_{\Delta} \quad = \quad & \texttt{vtx\_cost}(10 \cdot \left(\frac{s}{2}\right)^2) + \\ & \texttt{rst\_cost}(d^2, \min l, \sin\max\alpha) \end{aligned}$$

This formula is derived by adding the rendering costs for vertex processing (first row) and pixel rasterization (second row). The number of processed triangle fans of a mesh of size $(s+1) \times (s+1)$ is $\left(\frac{s}{2}\right)^2$. Each triangle fan consists of 8 triangles, which are rendered by processing a total of 10 vertices. When looking onto the terrain straight from above the entire area of the block accounts for rasterization. For non-orthogonal viewing angles we are weighting the area of the block with the sinus

of the maximum angle of view, which is a sufficient approximation for the projected area of the triangle mesh. Likewise, the terrain slicing costs for x-, y- and z-slices are given by:

$$\begin{aligned} \texttt{cost}_\texttt{y} &= m_y \cdot \\ &\quad (\texttt{vtx\_cost}(4) + \\ &\quad \texttt{rst\_cost}(d^2, \min l, \sin\max\alpha)) \\ \texttt{cost}_{\texttt{x/z}} &= m_{x/z} \cdot \\ &\quad (\texttt{vtx\_cost}(4) + \\ &\quad \texttt{rst\_cost}(d \cdot h, \min l, \cos\min\alpha)) \end{aligned}$$

The number of processed vertices is four times the number of slices, since each slice is drawn by a quadrilateral, whereas the number of rasterized pixels depends on the projected area of the quads.

By analyzing the cost of terrain slicing we observe that by each level of quadtree descendance the number of drawn slices approximately doubles, whereas the rasterization area decreases by a factor that is determined by the shape of the bounding boxes. In the worst case the rasterization area might not be reduced at all. In this case the total rasterization cost is decreased by a factor of $C \approx \frac{\min l}{\min l + \frac{d}{2}}$, which is due to the reduction of the number of required slices in the furthermost child nodes. Since our subdivision criterion is merely a comparison of the terrain slicing cost of the actual node and the sum of the terrain slicing costs of its children, the quadtree traversal might be stopped before actually the optimum level is reached. For this to happen the condition

$$\begin{aligned} &\texttt{vtx\_cost} + \texttt{rst\_cost} < \\ &\quad c \cdot (2 \cdot \texttt{vtx\_cost} + C \cdot \texttt{rst\_cost}) \end{aligned}$$

must be fulfilled. An intuitive interpretion of this equation is that it cannot be true, if the ratio C is lower than a certain unknown value C', which can be controlled by the constant c ($c < 1$). The other way round, we can ensure that our cost minimization is correct for every node with $C < C'$ by choosing an appropriate value for c. This means that our minimization is correct for every node with at least a certain projected size. In practice, we have found $c = 0.9$ to be a reasonable choice. We have also found that the rendering performance differs only slightly for varying values of c, which indicates that the optimization is already close to the optimum.

As mentioned in section 3, a lower bound on the number of required slices cannot be calculated in some cases. Furthermore, the correct estimation of the rendering cost for a bounding box that intersects the sides of the viewing frustum is difficult to achieve. In these cases we simply decide to descend further down the hierarchy. For a bounding box that is completely invisible the rendering cost is assumed to be zero, because it is culled by the quadtree.

In Figure 3 a data set of Yukon Territory, Canada, with 1025×1025 grid points is displayed by means of our hybrid terrain rendering technique. The same terrain is shown in Figure 4, but the sliced parts of the terrain have been replaced with their bounding boxes. The geometry of the triangle fans has been made visible by setting the color of the center vertex to black. One can clearly see that our adaptive terrain slicing method is chosen in the distance and in flat areas. In the vicinity and for huge slopes the brute force attack was considered to be faster, since otherwise a large number of tall slices would have been rendered.

In order to show the terrain slices, by which the height field is rasterized, the accuracy of our algorithm was lowered in Figure 2. One can spot a bunch of vertical slices, but no horizontal slice bundle, because the point of view is already too close to the surface.

5 Eliminating Possible Glitches

In order to prevent misalignment of meshed triangles, the vertices of adjacent triangle edges are usually specified in the same order. Otherwise it might happen that some pixels on a triangle edge are omitted. Depending on the rasterization stage of a graphics accelerator this effect is more or less visible, but can be prevented by specifying the vertices in the correct order. The seamless alignment of rasterized quadtree nodes is a bit more complicated. Fortunately, we only have to pay attention to the horizontal slices, since the vertical ones already have sufficient overlap due to perspective reasons. In order to ensure this for horizontal slices as well, we just furnish the slices with a border that is at least one pixel wide. The width of the border in object space can be approximated as follows:

$$\texttt{width}_\texttt{border} = x'(1, \min l, 1)^{-1}$$

Similarly, the vertical terrain slices with a projected height of less than one pixel have to be adjusted, too. Otherwise we would be able to look through the surface. We circumvent this problem by extending the slices at the bottom and at the top by an amount of one pixel. The latter also betters the alignment of polygonal nodes with rasterized ones, The alignment of triangles and slices is entirely seamless when raising the slices by an amount of exactly half a pixel.

6 Results

The traversal depth of the quadtree is not very high compared to the accuracy of the rendered image. Therefore, the CPU is offloaded efficiently and most of the work is accomplished by the graphics accelerator. Table 5 lists some performance measurements for the Yukon Territory data set with different window sizes. With the size of the window increasing the terrain slicing approach becomes more and more costly, because the rasterized area of each bounding box is increasing, too. Therefore, the polygon rendering alternative is simultaneously becoming more and more attractive, which leads to an increasing number of rendered triangles. Compared to exclusive polygonal rendering with quadtree culling our method achieves a speedup of about 70% to 400% depending on the size of the window.

Hence, the main limiting factor for the maximum number of frames per second is the rasterization bandwidth of the graphics hardware and neither the size of the rendered mesh nor the speed of the main processor. This fact is illustrated by Figure 6, which shows the city center of Stuttgart, Germany, with a grid resolution of 2049×2049 split into 2×2 tiles. The achieved frame rates were only slightly lower than those of the Yukon Territory data set with a gris size of 1025×1025. Accordingly, the speedup was approximately 3 to 4 times higher. The same holds for Figure 7 showing a highly detailed fractal landscape. In Figure 8 the intensity of the image is correlating directly to the number of rasterized pixels. Due to adaptive terrain slicing the overraw is decreasing in flat areas and with increasing distance to the point of view.

For the Yukon Territory data set the quadtree consisted of 21845 nodes, which were consuming approximately one megabyte of memory. The terrain was represented by a 1024×1024 texture map with 4 components. For the texture 4 megabytes of dedicated graphics memory were allocated. The quadtree representation was derived by resampling the texture with a grid size of 1025×1025. Comparing the complexity of our algorithm and the data structures with currently existing terrain rendering approaches the memory consumption is one of the lowest reported. What is more, no dynamically changing data structures or sophisticated propagation schemes are required.

window size	**avg. fps**	**min. fps**
400×300	35.3	28.8
512×384	28.5	21.5
640×480	22.2	18.1
800×600	18.0	14.2
1024×768	15.1	11.0

window size	**triangles**	**slices**
400×300	68,736	11,941
512×384	93,952	14,832
640×480	151,296	13,445
800×600	209,152	13,296
1024×768	257,536	10,042

Figure 5: Performance measurements for the Yukon Territory data set with a size of 1025×1025 on a 900 MHz AMD Athlon (CPU load: $<24\%$) equipped with a NVidia GeForce2 MX SDR (vertices/sec: 9.5M, pixels/sec: 157M). For comparison, the performance of brute-force polygon rendering with quadtree culling is at the minimum 3.1 frames/sec and 8.9 frames/sec on the average. This corresponds to 585,856 and 2,101,250 rendered triangles, respectively.

7 Terrain Slicing Extensions

In this section we will discuss a variety of extensions that can either improve the visual quality or the speed of our terrain slicing approach. Additionally, we are giving examples for other possible application areas besides terrain rendering:

- On machines that support register combiners (namely the NVIDIA GeForce graphics accelerator family) antialiased terrain slices can be drawn by using a simple trick. Instead of using an alpha test to cut away the solid parts of

Figure 6: The city center of Stuttgart from above.

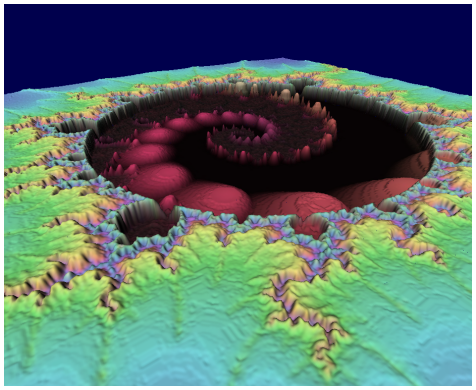

Figure 7: A highly detailed fractal snake.

the slices, the register combiners can be programmed to output opacity values, which decrease from one to zero, if a rasterized pixel is above the surface. Then the slices just need to be blended appropriately to obtain antialiasing.

- The NVIDIA GeForce3 graphics accelerator supports the vertex shader extension, which can be used to offload the CPU from calculating the vertex positions, texture coordinates and opacities of each slice in the bundle. The vertices of the starting slice of the bundle just need to be shifted along the axis of alignment. The opacities and texture coordinates of the slice quads can be derived directly from each 3-space vertex.
- Sorting the slice bundles and the quadtree in a back to front fashion allows the display of height dependent fog. As the alpha component is already reserved for the heights of the terrain, the opacity of the slices needs to be encoded into the RGB channels of the primary color with blending function `glBlendFunc(GL_SRC_COLOR, GL_ONE_MINUS_SRC_COLOR)`.

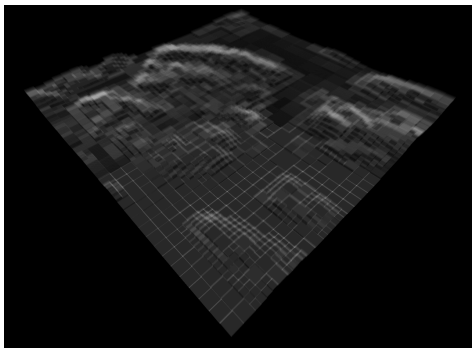

Figure 8: Yukon Territory with the image intensity correlating to the number of rasterized pixels. Brighter colors correspond to a higher overdraw.

- In order to improve the visual quality and to eliminate the angular look of triangles nearby one could apply adaptive subdivision surfaces to the polygonal mesh. Then the level of subdivision should depend on a view-dependent criterion.
- A hierarchical combination of terrain slices with traditional TINs is also worthwhile considering in order to reduce the number of rendered triangles. Keeping the error tolerance low, such that the maximum deviation of the irregular network falls below the height quantization error, provides better performance for smooth areas. A combination with continuous level of detail algorithms may also have synergetic effects. In such a hybrid approach our terrain slicing technique could be used to render distant parts of the terrain.
- Because of the simple data structures that were employed for terrain slicing, a partial modification or animation of a terrain is fairly easy to achieve. Basically, only the texture map and the bounding box hierarchy have to be updated each time the height field is changed.
- In interactive entertainment bump maps are commonly used to enhance the visual quality

of surfaces, but when approaching these bump maps they unfortunately reveal their flat nature. Introducing displacement maps at this point (see also [12]) helps to improve realism pretty much. To render these displacement maps only a small number of terrain slices are needed on the average, thus our approach is especially suited for displaying these maps at high frame rates (see Figure 9).

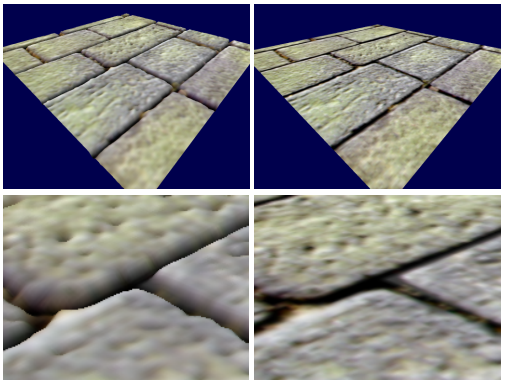

Figure 9: **Top Left**: A brick texture with a displacement map (minimum frames/sec: 32.5, average frames/sec: 49.3; for a distant point of view the maximum achievable frame rate is mostly limited by the vertical synchronization rate of the monitor). **Top Right**: The same brick texture, but with a flat bump map to show the visual advantages of displacement maps. **Bottom Row**: A part of the upper screen shots has been zoomed out to show the differences more clearly.

8 Conclusion

We have shown how to take advantage of the increasing rasterization performance of mainstream PC graphics accelerators in order to display height fields at interactive frame rates. We achieved this without compromising image quality. Our hybrid terrain rendering and slicing approach maximizes rendering performance by using a hierarchical bounding box representation of the terrain. Currently available graphics accelerators are not yet fast enough at high screen resolutions. Since the performance of our approach is mainly limited by the rasterization bandwidth of the graphics hardware, the point in time is predictable when high screen resolutions will be supported. Right now, however, our approach is well suited for the hardware accelerated display of displacement maps. Substituting the commonly used bump maps with the visually superior displacement maps can substantially enhance image quality in interactive entertainment.

Acknowledgements

We would like to acknowledge that this work was funded by the Ministry of Science, Research and the Arts of the State of Baden-Württemberg within the GISMO project. We also would like to thank Matthias Hopf, Martin Kraus and Marcelo Magallon for fruitful discussions.

References

[1] Daniel Cohen-Or and Yishay Levanoni. Temporal Continuity of Levels of Detail in Delaunay Triangulated Terrain. In *Proceedings of Visualization '96*, pages 37–42. IEEE Computer Society Press, October 1996.

[2] X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey. Multi-layered Impostors for Accelerated Rendering. In *Proceedings of Eurographics '99*, pages 61–73, 1999.

[3] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing Terrain: Real-Time Optimally Adapting Meshes. In *Proceedings of Visualization '97*, pages 81–88. IEEE Computer Society Press, October 1997.

[4] M. H. Gross, R. Gatti, and O. Staadt. Fast Multiresolution Surface Meshing. In *Proceedings of Visualization '95*, pages 135–142. IEEE Computer Society Press, October 1995.

[5] Hugues Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH '96*, pages 99–108. ACM SIGGRAPH, August 1996.

[6] Hugues Hoppe. View-Dependent Refinement of Progressive Meshes. In *Proceedings of SIGGRAPH '97*, pages 189–198. ACM SIGGRAPH, August 1997.

[7] Hugues Hoppe. Smooth View-Dependant Level-of-Detail Control and its Application to Terrain Rendering. In D. Ebert, H. Rushmeier,

and H. Hagen, editors, *Proceedings of Visualization '98*, pages 35–42. IEEE Computer Society Press, October 1998.

[8] Aleks Jakulin. Interactive Vegetation Rendering with Slicing and Blending. In *Short Presentations of Eurographics '00*, 2000.

[9] Jan Kautz and Hans-Peter Seidel. Hardware Accelerated Displacement Mapping for Image Based Rendering. In *Proceedings of Graphics Interface '01*, pages 61–70, June 2001.

[10] David Koller, Peter Lindstrom, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory Turner. Virtual GIS: A Real-Time 3D Geographic Information System. In *Proceedings of Visualization '95*, pages 94–100. IEEE Computer Society Press, October 1995.

[11] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. In *Proceedings of SIGGRAPH '96*, pages 109–118. ACM SIGGRAPH, August 1996.

[12] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief Texture Mapping. In *Proceedings of SIGGRAPH '00*, pages 359–368, 2000.

[13] Renato B. Pajarola. Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. In D. Ebert, H. Rushmeier, and H. Hagen, editors, *Proceedings of Visualization '98*, pages 19–26. IEEE Computer Society Press, October 1998.

[14] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage-Rasterization. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '00*, pages 109–118,147. Addison-Wesley Publishing Company, Inc., 2000.

[15] Stefan Röttger, Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel. Real-Time Generation of Continuous Levels of Detail for Height Fields. In V. Skala, editor, *WSCG'98 Conference Proceedings*, pages 315–322, Plzen, Czech Republic, February 9–13 1998.

[16] Gernot Schaufler. Per-Object Image Warping with Layered Impostors. In *Proceedings of the 9th Eurographics Workshop on Rendering '98*, pages 145–156, Vienna, Austria, June 29–July 1 1998.

[17] Sim Dietrich. NVIDIA white paper: Elevation Maps, `http://www.nvidia.com`, 2000.

[18] David C. Taylor and William A. Barrett. An Algorithm for Continuous Resolution Polygonalizations of a Discrete Surface. In *Proceedings of Graphics Interface '94*, pages 33–42, 1994.

[19] Rüdiger Westermann, Ove Sommer, and Thomas Ertl. Decoupling Polygon Rendering from Geometry using Rasterization Hardware. In D. Lischinski and G. W. Larson, editors, *Rendering Techniques '99*, pages 45–56. Springer-Verlag, Wien, New York, 1999.

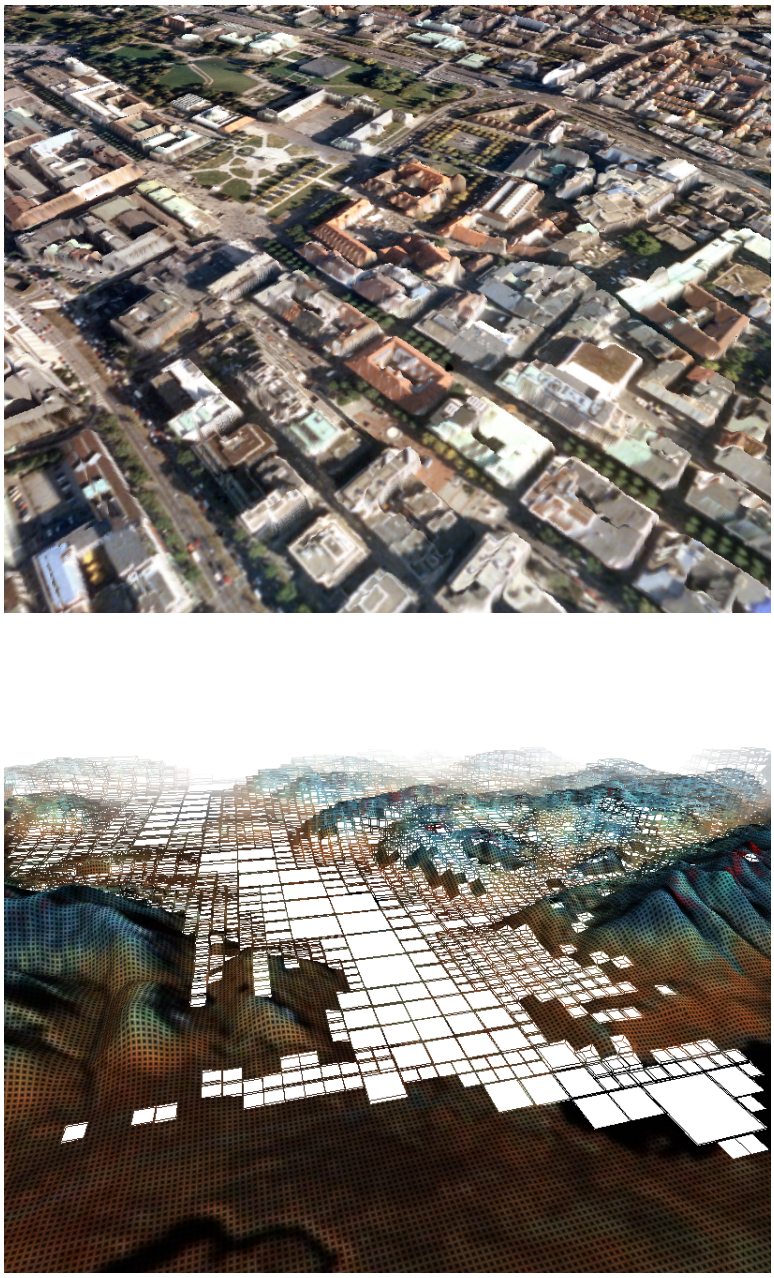

Figure 10: **Top:** The city center of Stuttgart rendered by using the hybrid terrain slicing technique. **Bottom:** A screen shot of Yukon Territory, Canada. The terrain slices have been replaced with their bounding boxes and the geometry of the triangle fans has been made visible by darkening the center vertex.