Error-bounded GPU-supported terrain visualisation

Falko Löffler
University of Rostock, Germany
Albert-Einstein-Strasse 21
18055, Rostock
falko.loeffler@uni-rostock.de

Stefan Rybacki
University of Rostock, Germany
Albert-Einstein-Strasse 21
18055, Rostock
stefan.rybacki@uni-rostock.de

Heidrun Schumann
University of Rostock, Germany
Albert-Einstein-Strasse 21
18055, Rostock
schumann@informatik.uni-rostock.de

ABSTRACT

The interactive visualisation of digital terrain datasets deals with their interrelated issues: quality, time and resources. In this paper a GPU-supported rendering technique is introduced, which finds a tradeoff between these issues. For this we use the projective grid method as the foundation. Even though the method is simple and powerful, its most significant problem is the loss of relevant features. Our contribution is a definition of a view-dependent grid distribution on the view-plane and an error-bounded rendering. This leads to a better approximation of the original terrain surface compared to previous GPU-based approaches. A higher quality is achieved with respect to the grid resolution. Furthermore the combination with an error metric and ray casting enables us to render a terrain representation within a given error threshold. Hence, high quality interactive terrain rendering is guaranteed, without expensive preprocessing.

Keywords: GPU-Rendering, terrain rendering, projective grid, level of detail

1 INTRODUCTION

The interactive visualisation of digital terrain datasets is a complex and challenging problem. Usually highly accurate terrain datasets contain billions of elevation and colour values, a data volume that cannot be displayed in real-time. View-dependent approximation of the terrain is needed to achieve interactive rendering.

In general, interactive terrain rendering has to address three interrelated issues:

- quality of the final image,
- restrictions regarding available resources, and
- the real-time capability of the algorithm.

The approximation of terrain data with respect to these criteria and for a given application context is a current research challenge. The problem can be characterised as follows: Usually we seek high quality. This can be accomplished either by spending more time on rendering or by storing pre-calculated results. On the other hand, we have to keep an eye on the resources used. Using fewer resources either leads to lower quality or might require to forego the real-time capability. Hence, changes with respect to one criterion necessarily affect the other criteria. The challenge is to find a good compromise between quality, used resources, and rendering time.

There has been extensive research on terrain visualisation. Today's algorithms can be categorised based on their utilisation of graphics hardware into *CPU-based* and *GPU-based* algorithms. CPU-based approaches focus on high quality and as such spend much time on complex calculations on the CPU. To achieve real-time capability they use pre-computed data structures that consume additional resources. However, the communication between CPU and GPU is often a transportation bottleneck that usually leads to lower frame rates. Moreover and inversely, most CPU-based terrain rendering algorithms use advanced error metrics, which directly affects rendering quality and time.

GPU-based algorithms, on the other hand, focus on real-time rendering by exploiting the parallel architecture of the graphics hardware. The idea is to perform many rather simple operations instead of a few complex ones to achieve high performance. This is possible through programmable vertex- and fragment processors of current graphics hardware. Even though GPU-based algorithms do not take the topology of the terrain into account, they can produce high-quality images due to the high primitive throughput. However, GPU-based algorithms usually cannot guarantee an approximation within a freely adjustable error rate.

All these terrain rendering approaches are powerful and well-designed. But some problems still exist in particular scenarios. For instance, CPU-based algorithms are not suited for resource-limited environments or for applications where the terrain is subject to modification during runtime. Vice versa, GPU-based algorithms are not the best choice in cases where a representation within a given error threshold is required.

Our approach focuses on a compromise between the competing needs for high quality, low resource consumption, and real-time capability. We have developed an algorithm that avoids expensive pre-processing, ensures real-time rendering, and achieves high quality within a guaranteed error threshold. This is particularly useful in aerospace systems, where resources are limited and a high quality visualisation is strictly required. Other applications like games can also benefit from our approach if they make use of dynamic terrain.

As the basis for our approach we use the *projective grid method* [13]. Johanson employs this method for real-time rendering of water surfaces that are modelled as dynamic height fields. The algorithm is easily portable to the GPU and suitable for very large terrain datasets using cliptextures [20]. Even though the algorithm can be applied for direct high-quality view-dependent rendering of height fields in real-time without any pre-processing, some problems can be observed. For instance, while navigating a height field, visual artifacts are recognisable. These are due to inadequate sampling and filtering of the height field. They are also caused by not taking the height field's topology into account.

In our approach, we reduce these visual artifacts to achieve high quality while maintaining real-time capabilities. Furthermore, we guarantee the terrain representation's quality within a given error threshold. To achieve real-time rendering, we employ a view-dependent sampling of the height field that results in a view-dependent level of detail (LOD) representation of the terrain. We use a GPU-tailored grid resolution for the sampling to fully exploit the power of the GPU. This leads to higher quality during rendering. Additionally, we generate an error map that gives us error boundaries for each elevation sample. In turn, the error map is used to control an adaptive ray casting that is applied to those regions of the image whose errors exceed a desired threshold. This way, the rendering quality can be guaranteed to be always better or equal to the given error threshold.

In the remainder of the paper we explain in detail how we achieve this good compromise between rendering quality, used resources and real-time capability. In the next section, we discuss approaches related to our work. In Section 3, we introduce the projective-grid method and discuss its major problems when applied to terrain rendering. Based on this discussion, we present and evaluate our own approach in Section 4. Section 5 is dedicated to the discussion of results.

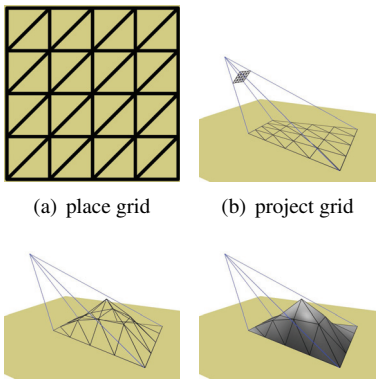
2 RELATED WORK

Terrain rendering algorithms can be categorised into *CPU-based* and *GPU-based* approaches.

CPU-based approaches construct, manage and select a proper approximation of the terrain data set using the CPU and the RAM. This allows utilising complex data structures and operations to construct terrain geometry. The composed geometry is then sent to the graphics hardware for rendering, which is often a bottleneck. The geometry of digital terrain data sets is usually described by triangles, which are directly supported by graphics hardware. Assembling a triangle mesh with regard to a sufficient triangle count leads to a good approximation of the terrain, provided that a proper triangulation algorithm is used. However, such meshes must be reassembled each frame to get a suitable view-dependent refinement of the original terrain data set. For example, [7, 24] apply a delaunay triangulation to limit the triangle count. This is also useful to improve the refinement and simplification of the terrain mesh and to reduce temporal aliasing. Whereas some approaches like [11, 12] do not constrain the triangulation process, other do so to generate and display hierarchies with multiple levels of detail. Many approaches use a regular network or quad-tree decompositions resulting in specialised and limited level-of-detail hierarchies. [17, 10] use binary trees to efficiently traverse and store the triangle hierarchy. Quad-tree triangulation is preferred by [2, 22]. The subdivision scheme from [19] subdivides the longest edge of a triangle to refine the terrain mesh. All these approaches extract a mesh on each frame, which restricts geometry caching and makes it difficult to utilise specialised techniques for efficient rendering. To solve this problem, [16, 22, 23] aggregate triangles to patches of different resolutions. At rendering time, patches of suitable resolutions are chosen to be combined and sent to the GPU. Hence, using patches accelerates the communication between CPU and GPU, but does not solve this problem entirely. Algorithms like [3, 4, 5, 26] store the patches in the graphics hardware's video memory. This significantly reduces data transmissions between CPU and GPU and hence increases rendering speed.

GPU-based approaches delegate the geometry processing to the GPU. These algorithms perform many simple operations rather than a few complex ones to achieve high performance through the parallel architecture of the GPU. In [1, 6, 9, 15, 14, 21] approaches are presented that can be implemented on today's programmable GPUs. A progressive geometry transmission is applied in [26] to reduce CPU to GPU communication. Warping and resampling of the underlying grid according to the viewpoint is done in [8]. This approach also adds procedural detail after resampling. Most GPU-based approaches use static levels of detail: the stitching of different resolutions is a common problem.

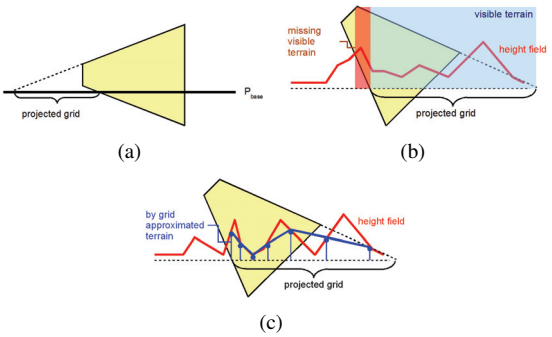
Another alternative for height field visualisation is the projective grid method. The method was first introduced by Johanson in [13] and was later applied to dynamic height field visualisation. Livny applied



(a) place grid (b) project grid

(c) displace grid points (d) use grid for rendering

Figure 1: Steps of the projective grid method



(a) (b) (c)

a) backfiring projection when looking above the horizon
b) intersection of terrain data peaks with view frustum
c) undersampled terrain and resulting grid

Figure 2: Visual artifacts caused by the projective grid method

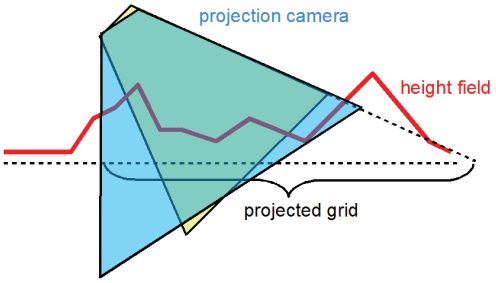


Figure 3: Projection camera with increased field of view to solve backfiring and intersected terrain

the approach to terrain rendering and combined it with clipmaps (see [27]) to support very large terrain datasets [20]. Instead of handling the geometry on the CPU, the grid is cached on the GPU and the programmable hardware is used to project and render the grid. This reduces CPU to GPU communication to a minimum. In [25], Schneider et al. use the projective grid method to display theoretically infinite terrain in high detail. Instead of precalculating height fields, they are generated at runtime.

Whereas Schneider et al.'s approach can not be used for predefined height fields, Livny does not guarantee rendering quality within a given error threshold.

We extend the projective grid method of Johanson in such a way that it is applicable to arbitrarily predefined or dynamic height fields. Furthermore we also ensure rendering quality within a given error threshold.

3 PROJECTIVE GRID METHOD

In this section we give a brief overview of the idea behind the projective grid method and describe the problems to be solved for its application to terrain rendering.

Basic idea

The projective grid method has been developed for interactive water rendering based on a dynamic height field. The principle of this method is simple and powerful. The basic idea is to cover the currently visible area of a height field and just this area, with a grid of fixed size which is placed onto the view plane. The size of the grid determines the quality of the terrain approximation and can be adjusted with respect to the capabilities of the used graphics hardware. The grid is projected onto the terrain's ground plane. Each projected point of the grid is displaced in the direction of the ground plane's normal by a fetched height value. The resulting grid is a view-dependent approximation of the original height field and can be used for rendering (see Figure 1).

Problem discussion

Even though the grid projection seems straight forward, there are three special cases which it needs to be adjusted in (see [13]):

- Looking above the scene's horizon results in *Backfiring*, which means that grid points will be projected behind the scene camera (see Figure 2(a)).
- In case of terrain data with *high amplitude*, peaks outside of the projected ground plane may intersect the view frustum (see Figure 2(b)).
- Undersampling can lead to a *loss of relevant features*, e.g., peaks and dips in the terrain. (see Figure 2(c)).

To solve the first two problems Johanson introduced the concept of an additional *projection camera*. This camera is aligned with respect to the viewing camera, but it never looks above the horizon. Moreover, to consider terrain that possibly extends into the view frustum, the projection camera's field of view is increased (see Figure 3). The problem of losing relevant features is not addressed by Johanson, because it can be ignored when rendering water surfaces. However, when applying the projective grid method for terrain rendering this problem has to be solved.

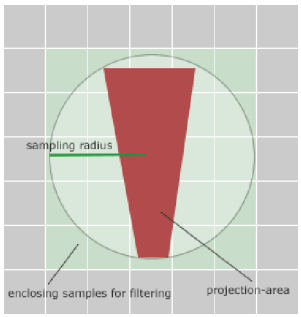


Figure 4: a non-uniformly shaped projection area leads to inadequate filter values due to the choice of the enclosing sampling radius.

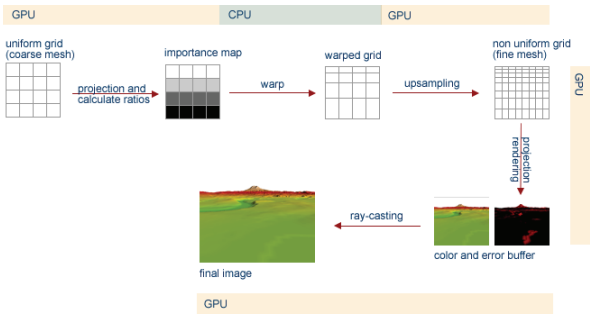


Figure 5: Scheme of our method's rendering process.

For this purpose, filtering of the height field has to be carried out. In [20], different resolutions of the height field are generated and the proper resolution depending on the sampling radius of a projected grid point is used. A drawback of this approach is that in situations where the view direction is close to the horizon, the projection of a single point in screen space onto the height field leads to a trapezoid area strongly elongated in the view direction, but narrow in the transverse direction (see figure 4). Because of the enclosing sampling radius used to determine the LOD, a filtered elevation value is chosen that does not approximate the underlying height field in a proper manner.

Undersampling as well as inaccurate filtering of the height values lead to a loss of relevant features, depending on the current view parameters and the resolution of the grid.

4 OUR METHOD

In this section we present our algorithm for interactive terrain rendering that addresses the problems described in the previous section. The general procedure can be described as follows: First, we generate a sample grid whose resolution depends on the capabilities of the graphics hardware. Thereby, we can guarantee the highest quality that is possible with respect to a given output device. Like in [20], we cache the grid in video memory, thus projection and rendering can be performed on the programmable graphics hardware. In contrast to previous approaches, we define the grid on the view plane depending on the current view in such a way that the projection of grid points results in a better approximation of the original terrain surface. This alleviates undersampling problems and helps achieve better image quality with respect to a given grid resolution. The projection is performed in a straight-forward manner. But contrary to known approaches, we compute an approximation error for each grid point using an extended MipMap hierarchy for the height field. The error values are used to generate an *error buffer*. During rendering the buffer is deployed for an adaptive per-pixel displacement mapping in regions where the error threshold is exceeded. This guarantees a representation within a given error threshold. A scheme of the rendering process is shown in figure 5. In the following, we will discuss the individual steps in more detail.

Grid Definition

The grid definition is a crucial step of the projective grid method. An accurate approximation of the original terrain surface implies a proper grid point distribution on the view plane. Earlier approaches used a fixed, pre-defined grid point distribution, leading to visual artifacts in particular situations (see Section 3).

These artifacts occur due to the fact that the projected grid points do not correspond to the original grid points of the terrain data. To alleviate this problem, we use a non-uniform, view-dependent grid point distribution. The grid points are defined in the view plane in such a way that the projection of the grid leads to almost quadratic grid cells. Thus, stretched grid cells caused by specific viewing conditions are avoided. This implicates that the region of influence of a projected grid point is also almost quadratic. As a result, artifacts caused by inadequate filtering are reduced. However, finding a good distribution is not a trivial task, because we need knowledge about the projection and perspective distortion. To define such a view-dependent grid point distribution in the view plane, a two step method is carried out:

First, a uniform grid is defined in the view plane and is projected onto the terrain's ground plane. The aspect ratio of each grid cell is calculated. This gives us a measure for the distortion of the grid cells. The aspect ratio is a sufficient measure, because it depends on the grid resolution as well as on the current view parameters. In the second step we use this measure to distort the uniformly distributed grid in the view plane, resulting in a non-uniformly distributed grid.

Whereas the first step is straight-forward, the second step can be implemented with the help of the importance-driven warping technique introduced in [8]. The warping function distorts the grid in such a way that more grid points are placed in regions

with high importance, while grid points are removed in other regions. This is exactly the behaviour that accomplishes our problem.

The required importance map is computed based on the aspect ratio of the projected grid cells. Regions with aspect ratios less than one are considered as very important, whereas regions with aspect ratios greater than one are declared as less important. This prompts the warping technique to relocate grid points from regions marked as unimportant to those declared as important. Hence, this results in the desired non-uniformly distributed grid.

This calculation is expensive and must be carried out on the CPU (see [8]) and therefore cannot be applied to the entire high-resolution grid. To reduce the calculation overhead, we use a coarse grid defined in the view plane. After applying the warping algorithm, we use the programmable GPU to refine the grid as far as possible with respect to the power of the graphics hardware.

Our procedure does not result in an optimal grid point distribution, but nonetheless, it leads to much better results than fixed, view-independent grid point distributions. Thereby, we are able to reduce visual artifacts and to achieve a better quality (see section 5).

Projection

After defining the non-uniform grid in the view plane, the grid points are projected using the algorithm introduced by Johanson. However, to reduce aliasing artifacts and to avoid a loss of relevant features, we calculate the height values of grid points with regard to their regions of influence on the ground plane.

To calculate proper height values, we filter the height field. We construct a multi-level texture pyramid of the height field, similar to [20], as follows: Starting from the original (finest) level, each level is constructed from the previous one by applying an average filter followed by halving its size in each dimension. The algorithm determines the level in the pyramid which a value is selected from depending on the region of influence. Similar to previous approaches, we calculate the farthest distance *dist* between adjacent projected grid points and use this distance to calculate the level in the texture pyramid as follows:

$$level = max(0, \log_2 dist) \tag{1}$$

In contrast to other approaches, our grid definition guarantees an almost uniform distance between adjacent neighbours of a grid point on the ground plane. This leads to more accurately filtered height values. The result is a better approximation of the original terrain surface (see Figure 6) with respect to the grid resolution.

Even though the projected grid could now be rendered using a simple texture mapping into the *colour buffer*, further enhancements are necessary to guarantee a high quality representation within a given error threshold.

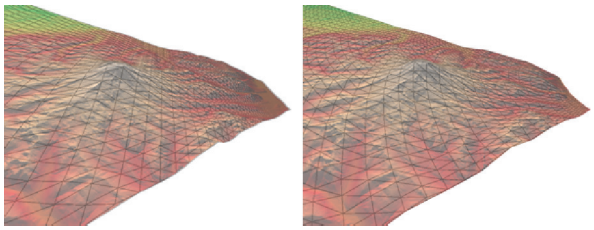


Figure 6: The left image shows the result of a uniform grid while the right image is generated using the view-dependent grid point distribution.

Error Metric

In our approach we want to guarantee a representation within a given error threshold. For that purpose, we use the following two error types:

- screen-space error
- object-space error

During the projection phase, the object-space error $\delta_{i,j}$ for each grid point $p_{i,j}$ is calculated. The object-space error depends on the chosen filtered height value h_{avg} as well as on the local minima h_{min} and maxima h_{max} in the region of influence of $p_{i,j}$. It is calculated as follows:

$$\delta_{i,j} = max(h_{max} - h_{avg}, h_{avg} - h_{min}) \tag{2}$$

To gather local minima and maxima we generate a *min* and *max* filtered texture pyramid similar to the previously generated average texture pyramid. In this way, average, min, and max height values can be fetched in unified manner from the texture pyramids. The fetching can be carried out in the projection step and the object-space error can be calculated using Equation 2. The object-space error is then projected back to the view plane, resulting in a screen-space error $\rho_{i,j}$. Since this can be computationally inefficient (see [18]), we use a simple metric:

$$\rho_{i,j} = \lambda \frac{\delta_{i,j}}{\|p_{i,j} - e\|} \tag{3}$$

with $\lambda = \frac{w}{\phi}$, where w is the number of pixels in the field of view ϕ and e the view position (see [18]).

The screen-space error $\rho_{i,j}$ can now be compared to the user-defined screen-space error threshold γ. If $\rho_{i,j} > \gamma$ we displace the grid point $p_{i,j}$ by h_{max} to preserve local maxima. Furthermore, the error is stored for each projected grid point $p_{i,j}$ and is used in the rendering pass to guarantee a representation within the error

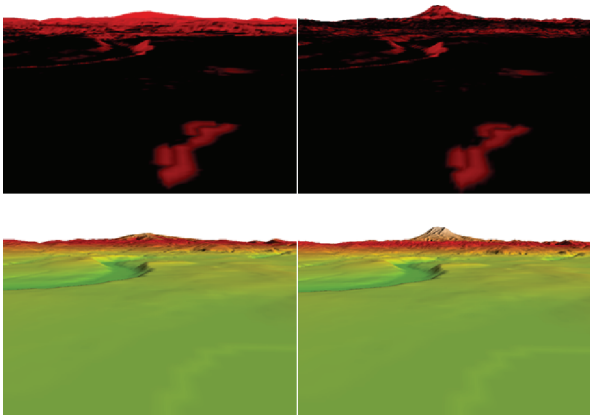


Figure 7: The error buffer for a 256x512 grid resolution. Red means high error, while black represents errors within the user-defined threshold. The left image shows the error buffer for a uniform grid point distribution. The right image was generated using the non uniform grid point distribution. Note the high detail and the minimised error in far-away regions.

threshold γ. For that purpose, we normalise the error values to the range $[0,1]$ as follows:

$$p_{i,j}.error = \begin{cases} 0 & \rho_{i,j} < \gamma \\ 1.0 - \frac{\gamma}{\rho_{i,j}} & else \end{cases} \quad (4)$$

Finally, the grid is rendered with the error value as colour attribute, resulting in an *error buffer* (see Figure 7) containing an interpolated error value for each visible pixel.

Rendering

During rendering our goal is to keep the per-pixel error below a given error threshold. Previous GPU-based approaches generated high quality images only by rendering huge numbers of primitives. But this does not guarantee any error rates. Therefore, we follow a different strategy. We perform adaptive ray casting in selected regions with errors that exceed the user-defined threshold. Hence, we are able to guarantee a chosen quality. The adaptive approach reduces calculation costs compared to applying ray casting to the entire height field. Ray casting is performed on the GPU as follows: For each pixel in screen space, the error is retrieved from the error buffer generated in the previous step (see Section 4). Ray casting calculates exact colour and precise depth values for a pixel in screen-space and replaces the less accurate ones in the colour and depth buffer (see Section 4). The final image can then be rendered using a *deferred shading* approach. We prefer deferred shading because it decouples shading from ray casting. Without deferred shading, to perform ray casting, we would require knowledge about the shading algorithm.

5 DISCUSSION AND RESULTS

Our approach can be summarised as follows:

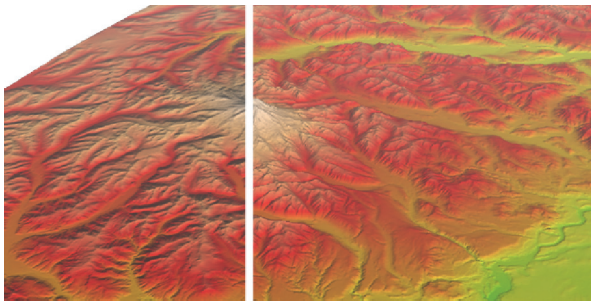


Figure 8: Ray casting of the terrain on selected areas. The left image shows terrain rendering without ray casting. On the right image ray casting is turned on.

The grid definition: defines a view-dependent, non-uniform grid on the view plane, which is novel compared to previous approaches. A uniform grid is warped with the help of an importance-driven method. The importance is defined by the aspect ratio of projected grid cells. This results in a non-uniform grid point distribution. Due to the view-dependent grid definition, we achieve a better approximation of the original terrain surface with respect to the grid resolution.

The projection: projects the non-uniform grid onto the ground plane and fetches proper height values for each projected grid point. The grid definition guarantees that the projected grid cells are almost quadratic, which leads to more accurately filtered height values. To avoid undersampling, the projection uses an average MipMap representation of the height field to fetch proper height values for each grid point.

The error measure: is used to gather approximation errors during the projection of grid points. In this step, a min and max MipMap representation of the height field is utilised. Based on the MipMaps, an object-space error is calculated for each grid point. The object-space error is projected back onto the view plane defining the screen-space error. The error is compared to a user-defined threshold and is normalised. An error buffer is rendered containing the interpolated normalised errors for each visible pixel.

The rendering process: performs adaptive ray casting utilising the error buffer in regions with high errors. The ray casting approach guarantees a representation within the user-defined error threshold.

The MipMaps reduce calculation time during the different steps. They can be generated in an offline process, but it is also possible to execute this during runtime, because the calculations are very simple and fast. Ray casting allows for a representation with a per-pixel error below a given error threshold. In fact, this can not

	fps				error	
grid size	uniform	non-uniform	ray casting (uniform)	ray casting (non-uniform)	uniform	non-uniform
1024x512	68.72	61.48	33.43	36.42	0.20	0.03
512x256	251.34	217.82	64.26	70.17	0.21	0.06
600x600	98.14	90.69	41.05	43.64	0.20	0.05
300x900	131.47	119.53	43.27	45.81	0.19	0.04
400x1900	48.61	45.80	23.52	25.34	0.18	0.02

fps: average frames per second for 8000 frames
error: average normalised error per grid point

Table 1: Speed and quality comparison between the standard method from [20] and our technique using different grid resolutions.

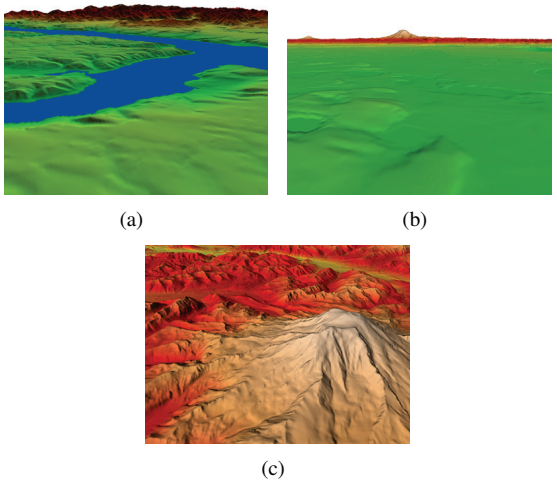
guarantee a fixed frame rate, as the original approach, but it is a good compromise between quality, time and resources. Indeed, the generation of the MipMaps consumes resources, but on the other hand, it enables us to guarantee a representation's quality. High quality is guaranteed by performing adaptive ray casting on selected areas, which, however, consumes time. But we keep ray casting to a minimum, by using an improved non-uniform grid point distribution on the view plane. This distribution is computed by a CPU-based warping technique, which again consumes time. However, except for the warping technique, all other calculations are performed on the GPU, which guarantees real-time and high quality terrain visualisation.

Our approach has been implemented using OpenGL 2.0 and requires graphics hardware supporting shader model 3.0 or higher. We use the vertex shader to define the grid on the view plane as well as for the projection and displacement of the grid points. The programmable fragment pipeline enables ray casting on the GPU. For the purpose of evaluation, we use the real-world 4k Puget Sound data set provided by Lindstrom with the original scaling factors having a peek at mount Rainier with ca. 4.400 metres.

It is also possible to support very large terrain using clipmaps as presented in [20]. Since Livny's and our approach use the same projection procedure, only a few modifications would be necessary.

The results we report in this section have been achieved on a PC with a Core 2 Duo 2.0 GHz processor, 1GB of memory and a GeForce 8800 GTX graphics card. Table 1 shows average frame rates (fps) as well as the average screen-space error per grid point, during a flight over Puget Sound with and without ray casting turned on (see figure 9). We tested various grid resolutions using the standard method and our technique, with a fixed screen size of *1024 x 800*. The uniformly distributed grid rendering corresponds to Livny's approach (see [20]).

As Table 1 shows, the usage of a non-uniform projection grid leads to a better approximation of the underlying terrain and reduces the average screen-space error per grid drastically. For instance, using a low grid resolution like *512x256* and a non-uniform projection grid generates an average error of *0.06* where a uniform grid with a four times higher resolution with *1024x512* still generates an average error of *0.20*.



(a) (b)

(c)

a) the start of the flight
b) near ground in the middle of the flight
c) close up at the end of the flight

Figure 9: The flight over Puget Sound. We tested various camera perspectives, from flight near ground till closeups.

	error pixels		avg error		max error	
grid-size	uniform	non-uniform	uniform	non-uniform	uniform	non-uniform
1024x512	10.7	7.5	1.60	1.42	18.0	6.77
512x256	15.6	12.1	1.70	1.40	23.1	6.9
600x600	11.9	8.5	1.83	1.53	22.0	8.3
300x900	13.0	9.0	1.68	1.44	19.3	6.7
400x1900	11.0	7.4	1.67	1.51	14.9	6.7

error pixels: the number of error pixels in % of all visible pixels
avg error: average screen-space error of all visible pixels
max error: max screen-space error of a all visible pixels

Table 2: Statistics on the errors of visible pixels. Our method minimises the regions, which ray casting must be performed in. Thus, we reduce the calculation time to achieve a representation within a defined error threshold. For performance issue see Table 1.

Comparing the frame rates of the standard method with our approach the time needed for warping is recognisable when using low grid resolution. The higher the grid resolution is, the more the frame rates converge. Looking at the grid resolution *400x1900*, the frame rate difference between the standard method and ours is very small and can be neglected.

Table 2 displays the percentage of error pixels in relation to the screen resolution (corresponding performance measurements are shown in Table 1. These regions must be handled by ray casting to guarantee rendering quality within the error threshold. Furthermore, the average error of all visible pixels as well as the maximum screen-space error have been captured. Comparing the maximum screen-space error of both techniques

shows that our technique approximates the original surface much better. Moreover, our method also minimises the regions with high errors. Hence, a lower resolution can be chosen, which still results in nearly the same number of error pixels, in contrast to the original approach. For instance, a *600x600* grid resolution generates fewer error regions with our technique than a grid resolution of *400x1900* with the classic approach. The results of Table 1 and Table 2 show that a compromise between time, resources and quality has been achieved.

6 CONCLUSION

We have introduced a GPU-supported approach for terrain rendering, using the projective grid method. We have shown how to reduce visual artifacts caused by inaccurate filtering of height values. Furthermore, we gather approximation errors that help us determine regions that need to be rendered using adaptive ray casting. Ray casting guarantees a representation within a given error threshold. We see the scope of future work in improving the view-dependent definition of the grid distribution in the view plane. Moreover, ray casting should replaced by a GPU-based subdivision algorithm utilising the shader model 4.0. This algorithm can be controlled by the error metric, and can be processed during the projection step. This will also increase the performance.

REFERENCES

[1] A. Asirvatham and H. Hoppe. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.

[2] X. Bao, R. Pajarola, and M. Shafae. Smart: An efficient technique for massive terrain visualization from out-of-core. In *VMV*, 2004.

[3] A. Brodersen. Real-time visualization of large textured terrains. In Stephen N. Spencer, editor, *GRAPHITE, Proc. of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia 2005, Dunedin, New Zealand, November 29 - December 2, 2005*, pages 439–442. ACM, 2005.

[4] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, September 2003. Proc. Eurographics 2003.

[5] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Planet-sized batched dynamic adaptive meshes (p-bdam). In *IEEE Visualization*, pages 147–154, 2003.

[6] M. Clasen and H.-C. Hege. Terrain rendering using spherical clipmaps. In Beatriz Sousa Santos, Thomas Ertl, and Ken Joy, editors, *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization*, pages 91–98, Lisbon, Portugal, 2006. Eurographics Association.

[7] D. Cohen-Or and Y. Levanoni. Temporal continuity of levels of detail in delaunay triangulated terrain. In *VIS '96: Proc. of the 7th conference on Visualization '96*, pages 37–42, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[8] C. Dachsbacher and M. Stamminger. Rendering procedural terrain by geometry image warping. In *Rendering Techniques 2004 (Proc. of Eurographics Symposium on Rendering)*, pages 103–110, 2004.

[9] W. de Boer. Fast terrain rendering using geometrical mipmapping. %urlhttp://www.flipcode.com/tutorials/geomipmaps.pdf, October 31 2000.

[10] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *IEEE Visualization '97 (VIS '97)*, pages 81–88, Washington - Brussels - Tokyo, October 1997. IEEE.

[11] J. El-Sana and A. Varshney. Generalized view-dependent simplification. *Computer Graphics Forum*, 18:83 – 94, 1999.

[12] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *VIS '98: Proc. of the conference on Visualization '98*, pages 35–42, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[13] C. Johanson. Real-time water rendering - introducing the projected grid concept. Master's thesis, Lund University, 2004.

[14] Y. Kryachko. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.

[15] B. D. Larsen and N. J. Christensen. Real-time terrain rendering using smooth hardware optimized level of detail. *Journal of WSCG*, 11(2):282–9, feb 2003. WSCG'2003: 11th International Conference in Central Europe on Computer Graphics, Visualization and Digital Interactive Media.

[16] J. Levenberg. Fast view-dependent level-of-detail rendering using cached geometry. In *VIS '02: Proc. of the conference on Visualization '02*, pages 259–266, Washington, DC, USA, 2002. IEEE Computer Society.

[17] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH '96: Proc. of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118, New York, NY, USA, 1996. ACM.

[18] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *IEEE Visualization*, August 2001.

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

[20] Y. Livny, N. Sokolovsky, T. Grinshpoun, and J. El-Sana. A gpu persistent grid mapping for terrain rendering. *Vis. Comput.*, 24(2):139–153, 2008.

[21] F. Losasso and H. Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph.*, 23(3):769–776, 2004.

[22] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In *VIS '98: Proc. of the conference on Visualization '98*, pages 19–26, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

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

[24] B. Rabinovich and C. Gotsman. Visualization of large terrains in resource-limited computing environments. In *VIS '97: Proc. of the 8th conference on Visualization '97*, pages 95–102, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[25] J. Schneider, T. Boldte, and . Westermann. Real-time editing, synthesis, and rendering of infinite landscapes on GPUs. In *Vision, Modeling and Visualization 2006*, 2006.

[26] J. Schneider and R. Westermann. Gpu-friendly high-quality terrain rendering. *Journal of WSCG*, 14(1-3):49–56, 2006.

[27] C. C. Tanner, C. J. Migdal, and M. T. Jones. The clipmap: a virtual mipmap. In *SIGGRAPH '98: Proc. of the 25th annual conference on Computer graphics and interactive techniques*, pages 151–158, New York, NY, USA, 1998. ACM.