Out-of-Core Terrain Rendering with Reparameterized Textures

Joachim Harabasz*

Institute of Computer Science II, Computer Graphics Group,
University of Bonn
Bonn / Germany

Abstract

Visualisation of terrain data has been an active area of research for more than an decade already and astonishing results have been achieved already. But there are still some issues to resolve, e.g. the texturing of current terrain rendering engines is based on orthographic projection, which leads to artifacts in very steep places like walls or cliffs, where small pieces of the texture is applied to large parts of the terrain model. To increase the texture information in these steep places it is necessary to increase the size of the texture area that is applied to these pieces of the terrain model. To solve the problem a new parameterization for the terrain model is computed and the texture is enhanced by additional information. Due to the new parameterization the orthographic texture has to be resampled to fit the new parameterization or in other words, the texture has to be "reparameterized". Ground-images are merged into the reparameterized texture to increase the texture details in steep places.

Keywords: Terrain Rendering, Reparameterized Textures, Image Projection

1 Introduction

In everyday life everyone is surrounded by some form of landscape. Modeling and rendering of this terrain is a much sought-after issue in many areas, like geographic information systems, space exploration and computer game development. In most cases an interactive visualization and a good image quality are desired. To achieve near photo realistic images a detailed terrain model is required. Recent improvements in remote data acquisition techniques allow to acquire datasets with a resolution of one height value per square meter even for large areas, with even more detailed texture data. This huge amount of data leads to big terrain models, which makes it difficult to achieve real-time frame rates in the rendering process. Wahl et al. [8] [9] created a terrain rendering algorithm which is capable of achieving frame rates of 150 fps even for huge terrain models in high-speed low-altitude overflights without concession to image quality.

Nevertheless textures acquired through satellite images or images taken through over-flights lack details in steep places. Steep places like house walls, cliffs and canyons are present in most mountain areas and nearly all man-made structures. This leads to highly noticeable artifacts in the rendered image, where a small piece of texture is stretched over a large piece of terrain model.

To increase the available texture information in these steep places, it is necessary to integrate information from additional sources like images taken from the ground. Applying this additional information can be done using additional projective textures. Unfortunately many of them will be necessary to achive sufficant quality which significantly affects the framerate of our terrain engine. Since we aspire to achieve real-time frame rates, this approach seems not practicable. We therefore decided to integrate all the additional texture information into the main texture. With an orthographic parameterization this is not possible, because the texture space in steep places will be very small.

To solve this problem, we decided to find a parameterization that respects area preservation, i.e. a parameterization that assignes texture space according to the surface area of parts of the terrain model.

For this the algorithm of Degener [4]. The new parameterization requires to resample the textures that were generated from satellite or over-flight images. These textures are referred to as "reparameterized textures" in this paper.

In general the used texture area will not be square shaped. To avoide a decrease in texture quality the texture size has to be increased in most cases. The reparameterization enables us now to add further details to the texture. This is done by projecting the additional terrestrial images that were taken from an arbitrary position and angle on the terrain model and merging the image data into the main texture.

First we will describe the principles of the rendering engine in section 3. In Section 4 we will explain the algorithm that is used to generate the texture coordinates. Subsequently in section 5 we will describe how the orthographic textures are adapted to the new parameterization,

*harabasz@cs.uni-bonn.de

and finally in section 6 how images are used to enhance the reparameterized textures.

2 Related Work

The method described in this paper is not related to a single area of research, but mainly combines results from the areas of terrain visualisation and parameterization methods.

In the area of terrain visualization much research has already been accomplished.

Among them some papers like Döllner et. al. [5] or Lindstrom et. al. citeLindstrom95 explicitly cover the topic of texturing terrain models , non of them tries to increase texture quality in case of steep terrain models. They mostly deal with topics of multi-resolution texturing of terrains mostly. Döllner et al. also discuss multi-textured terrains and the application of these techniques.

Many ideas have been developed regarding the fast rendering of terrain data. Some approaches, like the one of Cignoni et al. [2], are quite similar to the approach of our visualization engine described by Wahl [8]. They both use a set of triangulated irregular network (TIN) patches which provide the benefit that all operations can be performed on a per-patch instead of a per-triangle base.

Others like Lindstrom et al. [6] or Pajarola et al. [7] follow a view-dependent refinement approach, where a continuous terrain mesh is generated depending on the current view.

Although they achieve good results neither of them deals with the topic of texture parameterization and texture quality in steep places. Finding a "good" parameterization becomes normally harder with the size of the mesh. This makes [6] and [7] per se less suited for our goals. The patch based approach of [8] and [2] has the advantage that an independent parameterization can be calculated for every patch, which should lead to better results. Therefore choosing one of the patch based rendering engines for further enhancement regarding texture parameterization seems like a natural choice.

Similar to the area of terrain visualization, the topic of calculating good texture coordinates is not new and a lot of research has been done. A detailed overview over this research area can be found in [4].

Unfortunately a parameterization which preserves area, length and angle can not always be found. Though the most of the available algorithms calculate such a parameterization when it exists, they differ on the kind of compromises they make when it does not.

Although a lot of fine tuning can certainly be done by choosing a more appropriate algorithm, but we simply use the algorithm of Degener [4] since in our case any parameterization that is adapted to the terrain model will give better results than an orthographic parameterization.

3 The Rendering Engine

In this section the design of the rendering engine is described and we explain how the terrain data are structured to achieve a real-time framerates.

The rendering engine uses a quadtree based approach. A square cell of the terrain model is recursively subdivided into four sub-cells. Each sub-cell covers a fourth of the domain of the original cell and provides a more detailed version of the terrain model at this domain. In every quadtree cell a triangulated irregular network(TIN) is used. The quadtree is built in a preprocessing step. As input for this preprocessing step a height field is used. The quadtree cells are created in a bottom-up order. In the simplification process a conservative approximation of the Hausdorff distance is used as an error metric which directly corresponds to the screen space error. For each level of the quadtree an upper bound for the simplification error will be defined. Alongside with the geometry data in each cell, a texture and a normal map will be stored in constant resolution. This leads to an increase in texture data error by the factor 2 from one cell to a cell on the next higher level. Because of this, the error threshold for the simplification process is also increased by the factor 2 from one level to the next.

During the rendering process, depending on the view frustum origin and direction, quadtree cells can be chosen in such a way that their screen space error will not exceed 1/2 pixel. Thus the geometric error and the error in the chosen texture can be limited to 1/2 pixel. In this way popping artifacts, which originate from switching between different levels of details of the geometry or different texture resolutions can be avoided. All operations, like view frustum culling and occlusion culling, are executed per cell. They are executed by processing the bounding boxes data of the cells and thereby an unnecessary loading of the cell data into the main memory can be avoided. By feeding larger chunks of data to the graphic hardware, at least one cell, a better utilization can be achieved. Not every triangle has to be processed by the CPU.

Figure 1 shows part of the Puget Sound[1] data set[2]. With increasing distance from the camera position, larger cells are used. For each cell size the bounding box is drawn in a different color.

4 The Reparameterization

In this section the algorithm which is used to compute the new parameterization is described as well as the issues that come with this new parameterization. The new parameterization is necessary to increase the size of the texture regions that are applied to steep parts of the terrain model.

[1] Puget Sound is an area of Washington State USA.

[2] The Puget Sound data set is the largest terrain model current available to us. It contains 8 quadtree levels thus 21845 cells.

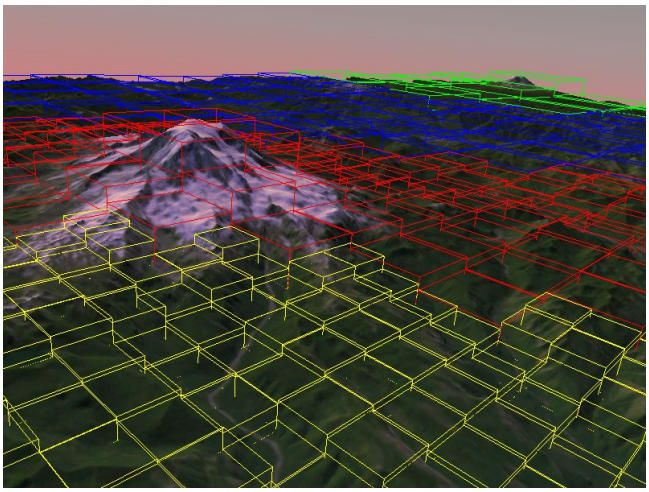


Figure 1: Image of the Puget Sound data set with bounding boxes for each cell.

The original terrain engine uses an orthographic parameterization. This parameterization is naturally associated with satellite or high altitude over-flight images when they are used for texturing. Using any different parameterization without the further enhancement of the texture leads mostly to lesser texture quality.

Choosing an area preserving parameterization also requires an increase of texture resolution, because the surface area of the terrain is larger than its orthographic projection. Keeping the old resolution would effectively mean to decrease the texture resolution for all parts of the terrain model, except for the steep ones. Also because of the new parameterization, the used texture area will not be perfectly square shaped any more. This means that the used texture area would be decreased without an increased resolution. Because most graphic hardware restricts the texture resolution to the power of 2, the next smallest texture size would thus mean a texture size increase by factor 4.

This also means that using an area preserving parameterization is only reasonable if the terrain model features areas with a rise of more than 60 degrees, because for a rise of 60 degrees an area preserving parameterization and an increase of the texture resolution by the factor 2 result in the same effect.

Increasing the texture size by a factor of 4 for all cells, means a significant increase of the total terrain model size. In natural terrains steep areas will appear in combination with very flat parts. For very flat terrain cells the new parameterization will mostly correspond to an orthographic parameterization. Thus the effect of the increased terrain area and the reduced used texture size will be very minimal. Therefore we analyzed how strong these two effects would be for an example terrain. For each cell we calculated the relative increase of terrain area that is covered by a texel Rel_{texel}. This is done by calculating the quotient of the terrain surface size of one cell over the cell base size and dividing it by the relative used texture size.

$$Rel_{texel} = \frac{surfacesize/basesize}{usedtexturesized/texturesize}$$

This formula calculates the average texture stretch for one cell. If the parameterization is area preserving, the average stretch will be identical to the stretch of each texel. It is not always possible to achieve an area preserving parameterization, but for flat areas this will mostly be well approximated.

A value of one indicates an optimal texture utilization. To increase the performance, it is reasonable to limit the texture size enlargement to cells that need it. Thus a threshold to characterize these cells is needed. We propose a threshold in the area of 1.1; this would limit the average stretching of a texel to 10%.

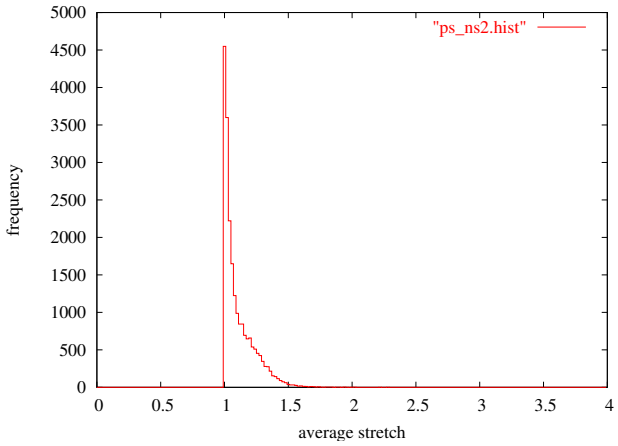


Figure 2: Histogram of the average texel stretch of the cells of the Puget sound data set.

Figure 2 shows the average stretch for the Puget sound data set. In this example 65% of all cells lie under the threshold of 1.1. In a more cliffy terrain model a worse ratio is to be expected, the Puget sound model is mostly quite flat.

The decision whether a new parameterization should be used or not, is independent of the question of the increased texture size. Thus only in cases where the steep area of a cell is very small, the use of area preserving parameterization without an increased texture size is reasonable. The use of an orthographic parameterization, were it is possible, is recommended. This way the unnecessary transfer of the texture coordinates to the GPU can be avoided. The mixed use of orthographic and area preserving parameterization has yet not been implemented because of time constrains.

4.1 The reparameterization algorithm

To compute the new parameterization we use an algorithm that was designed by Patrick Degener [4]. This algorithm can be applied to generate a parameterization for objects of

disc-like topology. Thereby it is selectable to what degree the area preservation is favored in opposition to the angle preservation, and vice versa. We have chosen to optimize both equally.

The algorithm can be subdivided into two major steps. First a simplification algorithm is used to simplify the object down to a single triangle. Second the simplification steps are reversed and for every vertex the texture coordinates are placed and optimized.

The simplification algorithm in the first step generates a Level of Detail structure, by generating lists of independent edge collapses. After each edge collapse the 1-ring around the resulting vertex is locked, and no edge collapse regarding one of these vertices is allowed. When no more edge collapses are possible, the locking of the vertices is released and a new edge collapse list will be generated. Because of the disc-like topology of the input object this process stops when a single triangle is reached.

For this single triangle trivially a parameterization can be found.

In the second step the lists of edge collapses are used to generate more detailed versions of the model, by applying the reverse operation of the edge collapse, the vertex split. For each vertex that is inserted an initial texture coordinate is computed. After a complete list of edge collapses is applied, a relaxation step is performed. In the relaxation step repeatedly a vertex is chosen and its texture coordinate is optimized regarding a special energy function. If no more improvement can be reached, the next list of edge collapses is applied. The whole process is also illustrated in figure 3.

This procedure avoids the problem that the relaxation process converges to a local minimum that only exists on a higher level.

For further details regarding this algorithm and the used energy function please refer to [4].

5 Resampling

In this section the resampling and filtering of the old orthographic texture is described, which is necessary to adapt the old texture to the new parameterization. A poor filtering in this step would lead to a texture quality decrease, which is contrary to the goal of increased texture quality (in steep places).

As well as the quadtree generation and the calculation of the texture coordinates the resampling is done in a preprocessing step. To avoid aliasing artifacts the resampling and filtering is mostly done in software. The fastest way would be to render the triangles of the geometry with the new texture coordinates from the reparameterization step as vertex coordinates and with applied texture, using the old texture coordinates for the texturing. This way the resampling could be done with fast hardware accelerated

[3]This figure is from [4] page 62.

Figure 4: Reparameterized and resampled texture of a cell of a small box-like terrain.

rendering, by using OpenGL [3]. But to avoid aliasing artifacts in the resampled texture, and to achieve an as good as possible texture quality we decided to do the filtering and resampling mostly in software, although we use hardware accelerated rendering to find the matching triangle to each texel of the reparameterized texture. But this results only in a speedup of 20 %. The most time consuming part of the resampling is still the filtering.

The filtering is done by approximating the intersection area of the texel of the resampled texture with the texels of the original texture by the use of subtexel. The final color of a texel of the resampled texture C_R is calculated as the weighted sum of colors of texel C_i that intersect with the footprint by the area A_i.

$$C_R = \sum_{i \in I} C_i * A_i / \sum_{i \in I} A_i$$

To increase the quality of the area estimation, the subtexels are evaluated inside a bounding box of the intersection area, of the footprint and the texel, also shown in figure 5.

In cases where the total intersection area $\sum_{i \in I} A_i$ falls below a threshold of 0.1 we fall back to bilinear filtering. This is mostly the case where the terrain makes vertical leaps, because in this case all four corners of the texel footprint will be collinear.

To reduce artifacts of applied texture compression algorithms, like S3TC or JPEG, the empty parts of the resampled texture are filled with an average border color. The average boarder color is an average color of filled texels of the texture which boarders on an empty texel.

Nevertheless, decrease in texture quality cannot be

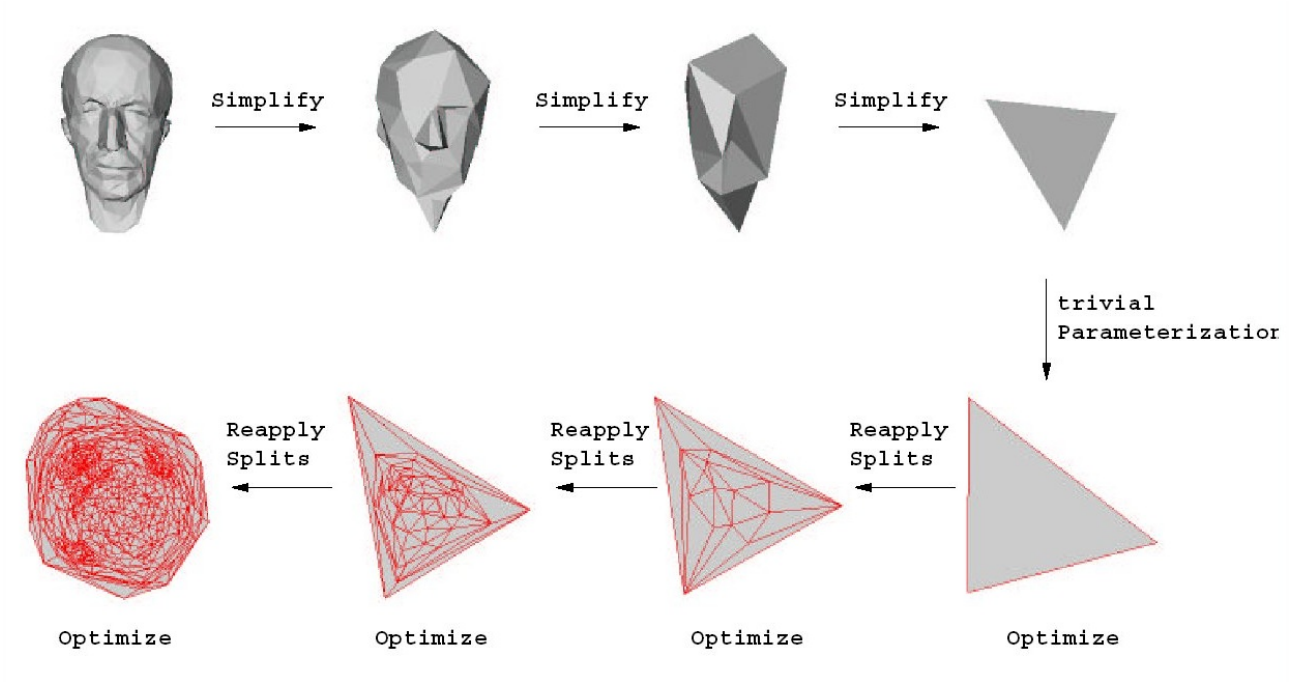


Figure 3: An overview over the process of texture coordinate calculation.[3]

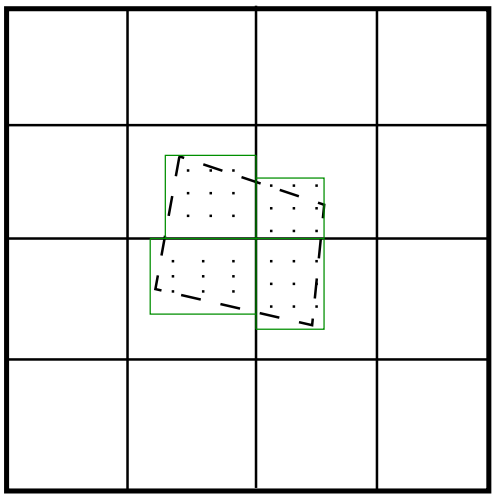
avoided in all cases, especially when the texture has strong features like a chessboard texture. In the reparameterized texture the texels are no longer aligned with the feature edges, which in case of a chessboard texture is quite noticeable.

Figure 4 shows a resampled texture for a small box like terrain also shown in figure 6. Due to the parameterization not the whole texture space is used. The texture area with the two paintings is applied to vertical parts of the box-terrain. The whole texture covers a fourth of the terrain. An image of this terrain is also given in figure 6.

6 Image Projection

In this section it is described how ground images are applied to the resampled texture, which is necessary to add texture information to the texture in steep places.

The process of image projection can be imagined as the reverse operation to the rendering process. As in the rendering process it is necessary to find the parts of geometry and texture that are projected to a specific pixel by the projection matrix. In opposition to the rendering process, not the texture color is written into the framebuffer, but the image color is written into the texture. To do this step correctly the position of the footprint of each image pixel is needed. This is done by calculating the texture coordinates for every corner of the image pixel. To calculate the texture coordinates a ray is cast through the corner of a pixel and the intersection point of this ray with a triangle is calculated. Now the barycentric coordinates b_i of the intersection point are calculated. By multiplying the



texel footprint

subtexel sampling point

Bounding Box of Footprint and Texel Intersection

Figure 5: Placing of the subtexels during the subtexel filtering.

barycentric b_i coordinates of the intersection point with the texture coordinates of the vertices that are incident to the triangle the texture coordinates t_i of the triangle are gained.

$$b_i = \frac{|(v_{i-1} - intersect) \times (v_{i+1} - intersect)|}{|(v_{i+1} - v_i) \times (v_{i-1} - v_i)|}$$

t_i = texture coordinate of vertex i

$$texCoord = b_1 * t_1 + b_2 * t_2 + b_3 * t_3$$

To speed up the process of finding the right triangle to an image pixel, we render an image in advance. Thereby a different color is applied to every triangle. This enables us to find the triangle with the nearest intersection point to the camera origin directly for every pixel of the image.

Figure 6 shows the effect of two images projected to the vertical sides of a box like terrain.

Figure 6: Image of a simple box-like terrain with enhanced textures.

Our rendering engine follows an out-of-core approach and the terrain model is only available in pieces of cells. To avoid an unnecessary increase in processing time, the loading of cells into main memory has to be minimized. Therefore a strict ordering of the processed cells in front to back order is required, where it is crucial that a later processed cell will not be able to occlude parts of a cell processed before.

6.1 Silhouette Pixel

When the right processing order is used, the cell occlusion will automatically be done corretly. But there are still some pixels which should not be written to the texture. This is the case where a pixel of the image covers a silhouette of the terrain model. A silhouette edge is an edge with an adjacent front-facing and back-facing triangle. If the pixel was written to the texture, the footprint of the pixel would be stretched over the texture space of a back-facing triangle, which leads to artifacts.

To decide if a pixel covers a silhouette edge or not, we check for each edge of the pixel footprint whether it crosses a silhouette edge in texture space. This is done by starting at the triangle of an end-point of a footprint border line and searching for all triangles that intersect with this line. If a back-facing triangle is found that intersects with a line, the pixel is treated as a silhouette pixel and is not drawn. To implement this we use an OpenMesh[1] data structure which allows us to access adjacent triangles of the mesh without searching.

In most cases this test can be avoided, because all four corners of a pixel correspond to the same triangle.

6.2 Footprint Reconstruction

In some cases the texture coordinates cannot be computed for all corners of a pixel. This is especially the case at cell boundaries, when the texture coordinates for some corners of the footprint lie in one cell and the others already in the next cell. Because every cell has its own texture and texture parameterization those cannot be used together. Not drawing those pixels would lead to a systematic error at the cell boundaries, therefore we decided to reconstruct the missing texture coordinates. If only one texture coordinate t_{ij} is missing, it can be computed from the remaining 3 by reconstructing a parallelogram.

$$t_{22} = t_{21} + t_{12} - t_{11}$$

When more texture coordinates are missing, they mostly can be estimated by using texture coordinates of the corners of adjacent footprints. In these cases it is important to check if a silhouette edge is crossed, because this could lead to extremely oversize footprints. We call this heuristic to calculate the missing texture coordinates of a footprint ”reconstruction scheme”.

In most cases the reconstructed footprint will be too large, because only a part of the pixel is projected onto the geometry. So the pixels will be written into parts of the texture where they should not be. But because the footprint reconstruction will mostly take place at the border of the cell, the oversize footprint will only cover the parts of the texture that were empty before. These empty parts are not mapped to any triangle. So writing to them cannot be considered an error.

On silhouette pixels no footprint reconstruction is performed, although the problem there is similar. But applying the footprint reconstruction scheme unchanged to the silhouette pixel would lead to falsely overdrawing used parts of the texture. This could be avoided by just drawing to the part of the texture that corresponds to a front-facing triangle, but this would just further increase the complexity of the process. Still it would be necessary to apply further images to fill the gap on the back facing triangles. It is likely that by applying those images, the gap that was left by the silhouette pixels will be filled. Thus we abstain from applying any reconstruction scheme on silhouette pixels.

In figure 7 and 8 the effect of the reconstruction scheme is shown. To increase noticeability we projected the image from a greater distance, though increasing the footprint size. In real application the effect should be smaller

but still noticeable.

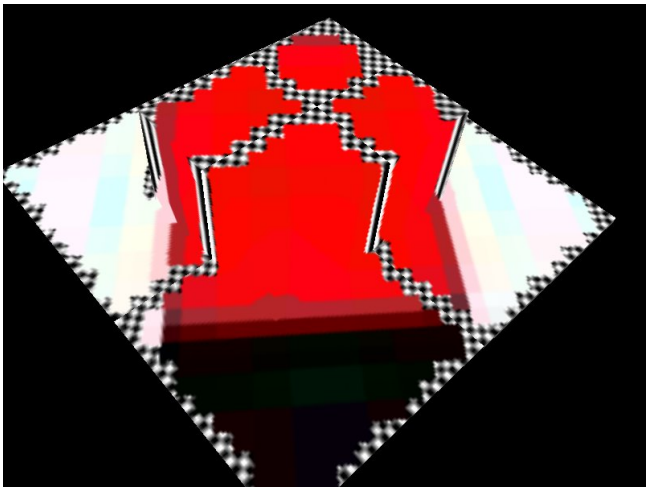


Figure 7: Picture of a box-like terrain with an image projected onto it. (without footprint reconstruction)

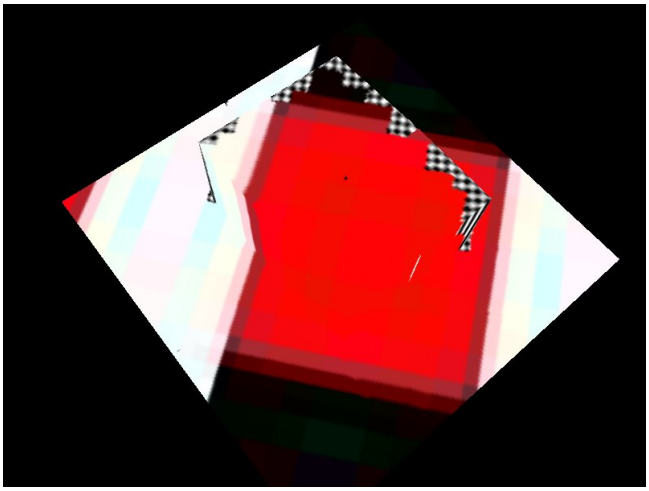


Figure 8: Picture of a box-like terrain with an image projected onto it. (with footprint reconstruction)

6.3 Filtering

The filtering in the image projection process is similar to the filtering described in the resampling process. This allows us to avoid aliasing artefacts, even when the resolution of the projected image exceeds the texture resolution by far.

7 Results

Figure 9 shows the performance of the modified terrain engine and the unmodified terrain engine using a high speed low altitude over-flight of the Puget Sound data set. As expected, the unmodified terrain engine outperforms the terrain engine with reparameterized textures. The difference in the reparameterized and unmodified 128 texel texture set results from the texture coordinates that additionally have to be transfered to the GPU. The unmodified terrain engine computes the texture coordinates within the GPU from the vertex coordinates.

The reparameterized 128-256 texel data set uses a texture resolution of 256 for all cells that have an average texel stretch of more than 1.1 . This is the case for 35% of all cells. The performance is only slightly lower than the performance of the reparameterized 128 texel terrain data set. The performance of the 256 texel data set is nearly identical to the mixed 128-256 texel data set. Thus the only benefit of the mixed resolution mode remains the size reduction of the terrain model by 45 %.

Figure 10 shows the performance of the unmodified terrain engine compared to the modified terrain engine once using double texture coordinates and once float texture coordinates. Noticeable is the extremely huge performance difference of the three different texture coordinate representation. The double texture coordinates are again 20 fps slower than the terrain engine with reparameterization and float texture coordinates.

The modified terrain engine is capable of rendering a terrain with reparameterized textures and only slight performance decrease, keeping the frame rate above a level of 30 fps most of the time, though keeping the frame rate above a real-time requirement of 25 fps.

Figure 6 shows a small box-like terrain, with two images projected onto the vertical walls. This would not be possible without a changed parameterization. We clearly achieve our goal of improving the texture quality in steep parts of the terrain model.

8 Future Work

It is still possible to enhance the performance of the terrain engine by implementing a mixed mode of orthographic and area preserving parameterization, thus minimizing the need to transfer the texture coordinates to the GPU.

Although we find that our proceeding is capable of increasing the texture information of terrain models in steep areas, usually a larger number of images will be required to achieve a satisfying result. The integration of larger numbers of images requires an automated process to acquire the camera position, direction and field of view. A manual estimation, which has been used until now, is not practical for more than very few images. Also a blending process that combines images projected from multiple positions and angles onto the terrain model, which considers the confidence of the projected image, will also be necessary to achieve a satisfying result. Thus we must state that more work is necessary to achieve a set of tools that enables us to enhance the texture of real terrains in steep places.

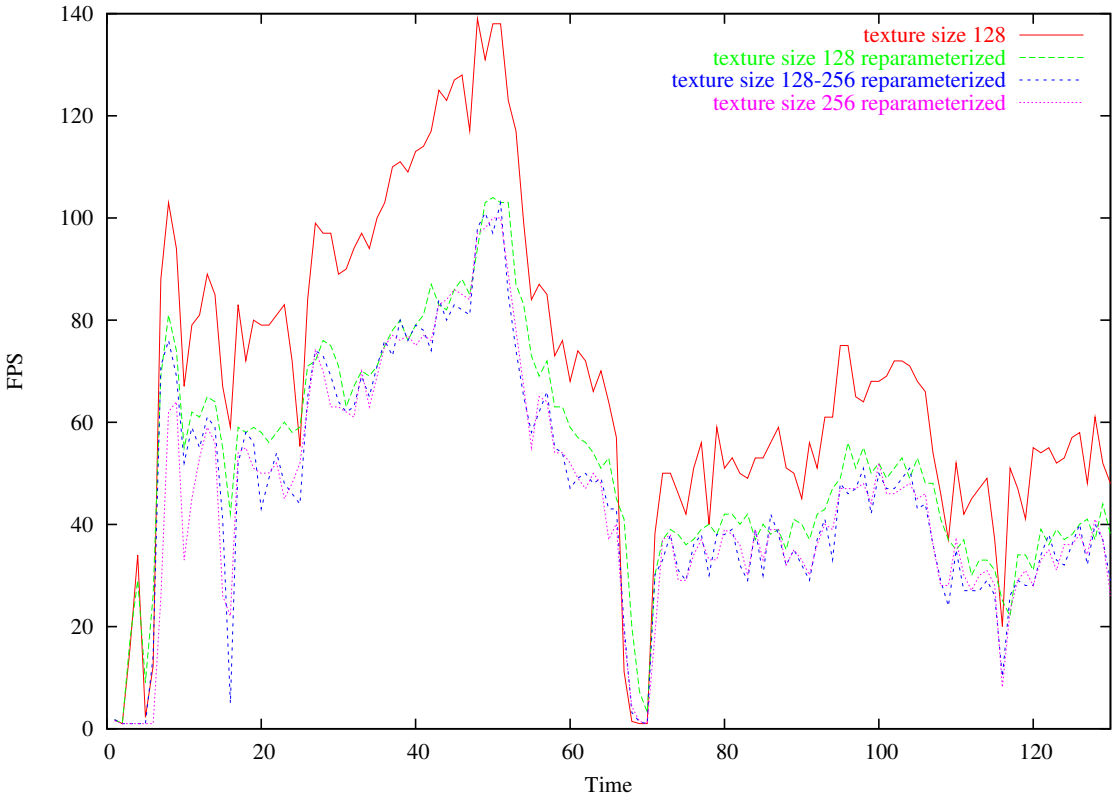


Figure 9: Performance comparison of different texturing methods and texture resolutions.

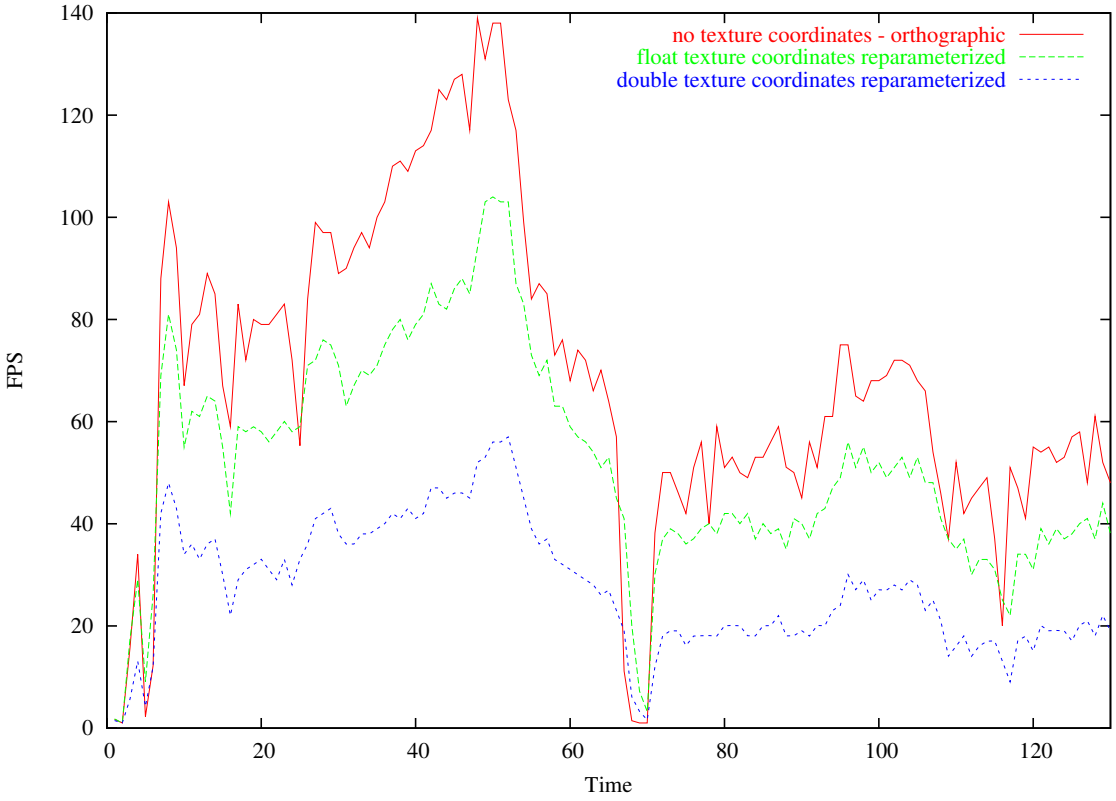


Figure 10: Performance comparison of different texture coordinate representations.

References

[1] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh – a generic and efficient polygon mesh data structure, 2002.

[2] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, and F. Ponchio. Bdam - batched dynamic adaptive meshes for high performance terrain visualization, September 2003.

[3] Tom Davis and Mason Woo. *OpenGL Programming Guide*. Addison Wesley, 1999.

[4] Patrick Degener. *Computing Parameterizations of Triangulated Surfaces with minimal metric Deformation*. Diplomarbeit, Institute of Computer Science II, Computer Graphics Group, University of Bonn/Germany, November 2003.

[5] Jörgen Döllner, Konstantin Baumann, and Klaus Hinrichs. Texturing techniques for terrain visualization. In *IEEE Visualization*, pages 227–234, 2000.

[6] Peter Lindstrom and Valerio 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, July–September 2002.

[7] R. Pajarola, M. Antonijuan, and R. Lario. Quadtin: Quadtree based triangulated irregular networks. In *IEEE Visualization 2002*, pages 395–402, 2002.

[8] Roland Wahl. *Scalable Compression and Rendering of Textured Terrain Data*. Diplomarbeit, Institute of Computer Science II, Computer Graphics Group, University of Bonn/Germany, December 2003.

[9] Roland Wahl, Manuel Massing, Patrick Degener, Michael Guthe, and Reinhard Klein. Scalable compression and rendering of textured terrain data. *Journal of WSCG*, 2004.