A Real-Time Physically Based Algorithm for Hard Shadows on Dynamic Height-Fields

Turgay Aslandere, Markus Flatken, Andreas Gerndt

German Aerospace Center (DLR), Simulation and Software Technology

Abstract: Bump maps are commonly used in computer graphics to visualize bumps and wrinkles on the surface of an object, more specifically, in height-field rendering. In order to render shadows for large terrain data using bump maps, various methods such as horizon mapping have been suggested. A horizon map describes the occlusion of terrain data by using angles with respect to a view point. In this paper, we propose a method exploiting ideas from horizon mapping to generate real time shadows for large terrain data. We only consider a single azimuth direction per height field point to compute the shadows. The generated horizon map is computed in real time and allows for interactive exploration of the terrain. We demonstrate that our method produces accurate results while keeping the memory requirements low.

Keywords: Height-Field Rendering, Horizon Mapping, Bump Mapping

1 Introduction

Height-fields are commonly employed in terrain rendering applications. They are not only used in games but also in real world applications such as flight and space mission simulators. In addition to real time rendering, such simulators require physically based methods to generate highly accurate shadows. These high quality shadows are absolutely necessary e.g. for the scientific simulation of camera sensor images used for the optical navigation of autonomous spacecraft landing vehicles.

Several physically based shadow algorithms have been investigated in order to render height-fields [Lai06]. One of the most common algorithms is ray tracing to produce the shadows. Although ray tracing can offer highly accurate results, it is computationally expensive [WSBW01]. Thus, more interactive shadow algorithms have been proposed such as shadow mapping [CD03] and horizon mapping [SC00]. These algorithms just provide shadow approximations. Shadow mapping techniques are interactive as they just approximate the shadows by testing whether a pixel is visible from the light source. However, the accuracy of these methods strongly depends on the depth buffer precision and shadow map resolution. In contrast, the horizon mapping method describes the visibility of a terrain in terms of angles with respect to a view point.

The main motivation of our presented work is to render realistic images of celestial bodies for scientific applications. Global shadowing is a crucial issue. In our approach, atmospheric scattering is ignored in a first step which is suitable e.g. for the Moon and asteroids. These

celestial bodies are generally rendered with a single light source (the Sun) and hard shadows. An additional challenge is the huge amount of sensor data from space missions which has to be used for the terrain rendering. To be fast but still accurate, Level of Detail (LOD) approaches has to be incorporated for the rendering. On the other hand, this yields new difficulties for an accurate shadow computation.

In this paper, we address this problem and present a physically based algorithm to guarantee real time rendering of large terrain data including the calculation of hard shadows. Our method exploits the ideas from horizon mapping [Max88]. The main contribution of this paper is that our method does not require pre-computed maps. Unlike the horizon mapping technique, we determine the visibility at run time. Hence, it has low memory requirements and avoids storing large additional data. Additionally, we do not rely on a number of fixed directions for the visibility check but use always just one azimuth direction. Therefore, our method does not introduce interpolation errors. In this case, it can be assumed as an optimized version of the horizon mapping. Moreover, our algorithm is applicable for Level of Detail (LOD) data structures.

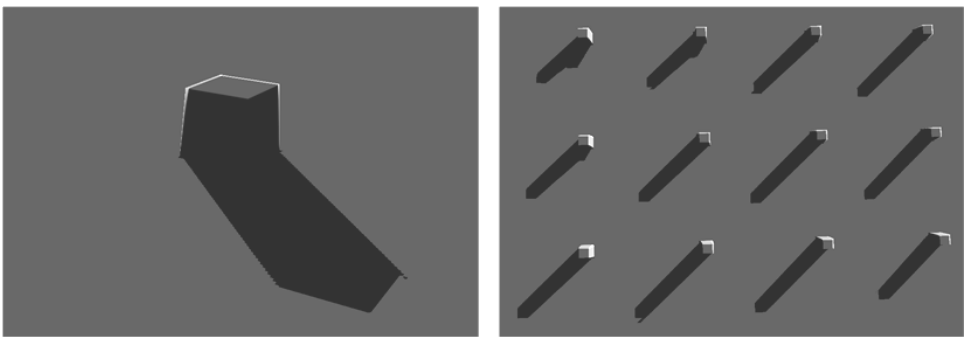
The next chapter gives an introduction to the related work in this field. Then, we explain the basic ideas and concepts. In the implementation chapter we describe our implementation. Later, we discuss the implementation issues and present that modern GPUs are capable to handle the computational complexity of the algorithm. Finally, a conclusion is given and possible future work is discussed.

2 Related Work

Bump mapping is a basic shading technique for rendering bumps [Bli78] on flat textures. It perturbs the mesh normals to modify the local shading. This can improve the perception of bumps. Because of the flat nature of the mesh, this technique does not consider any occlusions. Max presented horizon mapping [Max88] to overcome missing shadows on bump mapped surfaces. The main idea of horizon mapping is to pre-compute the visibility for each fragment of the bump map (height-field) and save this information in the horizon map. The visibility is determined by the slope which represents the highest elevation spot seen from a respective view point. For each azimuth direction (e.g., South, East, West, North), the angles to the horizon are encoded into discrete number of directions. Therefore, a horizon map is pre-computed for each defined azimuth direction. These horizon maps are used to interpolate and compute the occlusion along a used azimuth direction at run-time. Using this method, the interpolation errors will decrease with increasing pre-computed azimuth directions (Figure 1).

Synder et al. introduced soft shadows using a partial swath instead of applying a discrete number of directions for a full swath [SN08]. The partial swath only included 2 to 3 azimuth directions where the light comes from. They demonstrated effects of various interpolation methods (e.g., linear, b-spline) on the results.

Becker and Max depicted that various bump mapping algorithms (including displacement



(a) (b)

Figure 1: Artifacts that might occur because of horizon mapping (with 16 azimuth directions, linear interpolation between the discrete directions) where the azimuth direction does not match with the discrete directions. Shadowed areas are depicted in black. (a) A single cube on the height-field. (b) The same cube is duplicated 12 times on the height-field, the camera is located above the height-field.

mapping) can be blended [BM93]. Displacement mapping method does not just change the normals but displaces the surface itself. In addition to the work of Becker and Max, Wang et al. has developed a view-dependent displacement mapping method [WWT+03]. They also compared their results to horizon mapping and bump mapping.

Shinoyama and Max [SM10] demonstrated fast height-field rendering with image-based lighting. They approximated the environmental illumination using spherical radial basis functions (SRBFs) and employed horizon and cone mapping for their method. Additionally, their approach is capable to calculate self-shadowing.

Max [Max88] showed that horizon mapping is also capable of rendering shadows on curved surfaces. Onoue et al. proposed algorithms to compute bump map shadows by taking surface curvature into account [OMN04]. Their algorithm includes an additional distance map to optimize the algorithm for efficiency.

Nowrouzezahrai and Snyder presented a real-time rendering method for global illumination effects from large area and environmental lights on dynamic height-fields [NS09]. Their method requires no pre-computation for the horizon maps and is capable to reduce the noticeable artifacts of the horizon mapping method. On the other hand, is does not produce physically based shadows. And it does not deal with LOD data structures.

3 Basic Ideas and Algorithm

Our method is motivated by the horizon mapping technique. Unlike conventional horizon mapping, we only compute a single horizon map along the actual azimuth direction. Therefore, it requires similar parameters such as the maximum horizon angle (β), elevation angle (α), and azimuth angles (θ) to simulate hard shadows. θ and α on a flat surface are given by Eq. 1 and 2, respectively. The position of the light and the position of each pixel on the height-field texture is given by $L(x_L, y_L, z_L)$ and $P(x_P, y_P, z_P)$. z refers to the (height) value

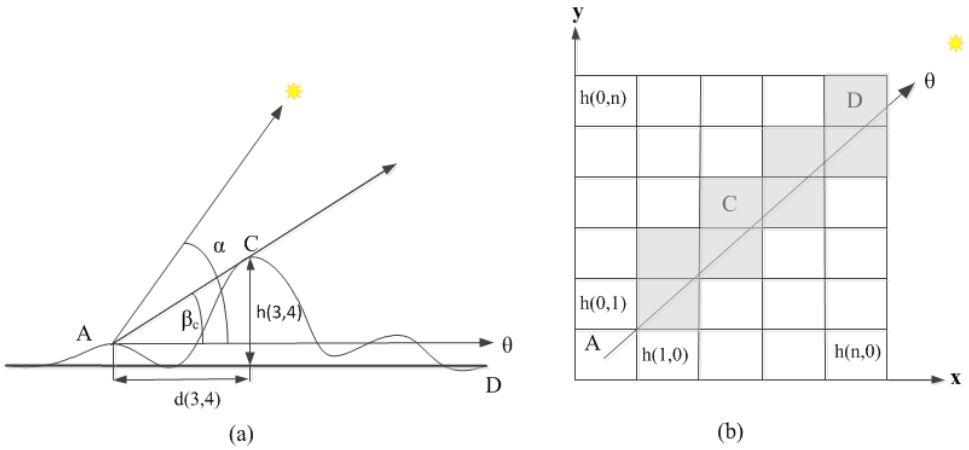


Figure 2: Horizon and elevation angles on the height-field. The size of the height-field texture is (n x n). (a) Relation of a point A and C on bump map. β_C represents the horizon angle at point C. (b) Point A and C on height-field texture. The colored pixels (samples) are used to compute maximum horizon angle at A. In this case, β_C is the maximum horizon angle.

of the pixel. Accordingly, the normal of the surface is $\vec{N} = (0, 0, 1)$ and the light vector is $\vec{L} = (x_L - x_P, y_L - y_P, z_L - z_P)$. Until now the light source is a point light but can also be defined as a parallel light source, such as the Sun.

First, the elevation and azimuth angles are computed for each pixel on the height-field texture. Thus, the surface normals $\vec{N}$ would be different for a curved surface.

$$\theta(x_P, y_P) = 90.0 - 180.0/\pi \; atan2 \; (x_L - x_P, y_L - y_P) \tag{1}$$

$$\alpha(x_P, y_P) = 180.0/\pi arcsin(\vec{N} \cdot \vec{L}) \tag{2}$$

Second, we compute the maximum angle attained by the horizon along the azimuth direction for each pixel on the height map. Unlike [SN08], we sample the height difference for a single light direction rather than a 2D swath. In Figure 2 (a), the elevation and horizon angle for a point A is depicted. We consider the height-field as a raster graphics image ($n \times n$) that evaluates the height information at a planar position (x_p, y_p). Height information can be represented by a scalar function $h(x_p, y_p)$ where $0 \leq x_p \leq n$, $0 \leq y_p \leq n$. Therefore, it returns a height value per coordinate pair x_p and y_p. $h(x_\theta, y_\theta)$ refers to height values towards the light source.

$$d(x_\theta, y_\theta, x_P, y_P) = \sqrt{(x_\theta - x_P)^2 + (y_\theta - y_P)^2} \tag{3}$$

$$\beta(x_P, y_P, \theta) = \arctan\left(max\left(\frac{h(x_\theta, y_\theta) - h(x_P, y_P)}{d(x_\theta, y_\theta, x_P, y_P)}\right)\right) \tag{4}$$

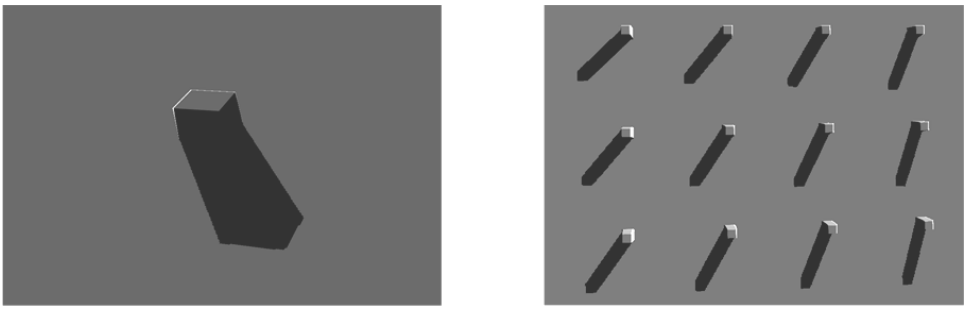
$$x_\theta = s\cos(\theta), \;\; y_\theta = s\sin(\theta), \quad where \; s \in (0, \infty) \tag{5}$$

Function d returns the distance between the sampling pixel and the pixel itself. Eq. 3 is given for a planar surface. The horizon angle is given by Eq. 4. We note that $h(x_\theta, y_\theta)$ are

only computed for the samples along the azimuth direction. The horizon angle would vary between 0 and 90 degrees.

Sampling methods play a major role for the accuracy. In Figure 2 (b), the samples to compute the horizon angle at point A are depicted. A continuous sampling can be achieved by Eq. 5. However, there might be other methods to sample the height-field along the azimuth direction. In this paper, we will not focus on the sampling methods.

The number of pixels that are taken into account during the computation can be reduced by sample spacing (sp). In order to achieve this, s in Eq. 5, $s \in \mathbb{Z}^+$, can be multiplied by a coefficient such as sp where $sp \in \mathbb{Z}^+$. This would speed up algorithms. For height-fields with high frequencies, it might decrease accuracy. Moreover, it may cause artifacts where $sp > 2$ due to the low number of samples. As a result, the sample spacing coefficient is taken as $sp = 1$. This would guarantee accurate results with stable performance. On the other hand, sp can be increased for height-fields with low frequencies depending on the requested accuracy. Therefore, a trade-off between accuracy and performance can be defined by the user. Furthermore, this trade-off is strongly dependent on the application.



(a) (b)

Figure 3: Hard shadows by our method. (a) A single cube on height-field. (b) The same cube is duplicated 12 times on the height-field. Our method does not introduce artifacts as conventional horizon mapping.

After determining the elevation and maximum horizon angle for each pixel, the occlusion of the pixels on the surface can be computed by the following function:

$$v(x_p, y_p) = \begin{cases} 1 & \beta(x_p, y_p) \leq \alpha(x_p, y_p) \\ 0 & \text{Otherwise} \end{cases} \tag{6}$$

$v(x_p, y_p)$ always returns true or false while false refers to a shadowed pixel. Our algorithm only computes hard shadows (Figure 3). However, Eq. 6 can be improved by methods suggested by Max for the horizon mapping [Max88]. In his case, the angles β and α can be employed to create penumbras for a circular light source. Moreover, the lighting can also be enhanced by adding more light sources.

4 Implementation

The rendering is based on the OpenGL 4.0 standard. Additionally, our implementation relies on CUDA 5.0 [nvi15] parallel computing platform to harness the power of graphics processors. And finally, we use the double precision floating point format provided by the Nvidia Quadro architecture in order to ensure the accuracy.

During the initialization phase of the rendering application, the height-field textures are loaded to the GPU memory via OpenGL. Figure 4 (a) shows a very simplified texture used for evaluation purpose. After it is uploaded, the rendering loop can start running. The height-field in the GPU memory is bounded to a CUDA texture exploiting OpenGL interoperability. Therefore, no host memory transfer is required anymore. The horizon map texture (Figure 4 (b)) is computed each frame for a given light source position (in our case: the moving Sun). In a first computation step, each GPU thread determines the horizon, elevation, and azimuth angle in parallel. Then, each thread compares the elevation angle to the horizon angle (visibility function). In an optimal case, a pixel in the horizon map is assigned to a dedicated GPU thread so that the computation is performed thoroughly concurrently. The result of the comparison is saved to the horizon map texture and passed on to a fragment shader which evaluates the horizon map values. If this visibility value is set to *false* (i.e. shadow), the color value in the frame buffer is set to black (hard shadow).

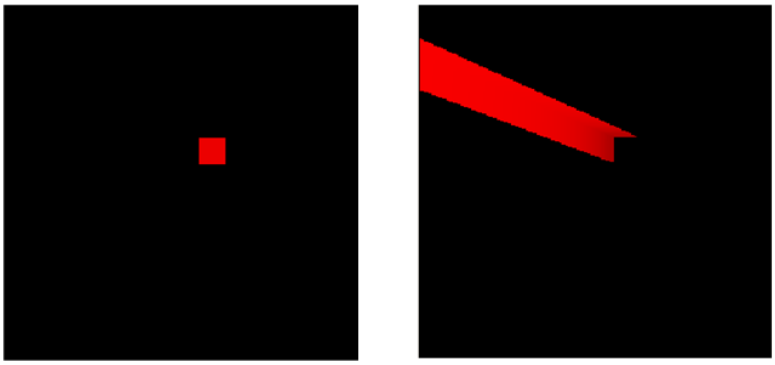


Figure 4: The textures in the GPU memory. (a) The height-field texture: An artificial height-field is depicted consisting of a single cube on a flat surface (cf. Figure 3 (a)). (b) The horizon map texture: The shadow values are marked in red.

Instead of visibility flags, it is also possible to store the horizon angle for each pixel in the horizon map. Actually, this is why [SC00] defined this texture as *"horizon map"*. They store horizon angles as floating point values which come along with higher memory consumption. In our algorithm, both approaches are implemented. However, we did not observe any significant differences in performance and accuracy.

We have also combined and extended our method to work with a Level of Detail (LOD) data structure [LKR+96]. Without involving such techniques, it would not be possible to process real space mission sensor data interactively. Therefore, in a preprocessing step, all

sensor sources and data stripes are mapped to quad-tree data structures. The leaf nodes contain the original data as grids which are considered as our height-field textures. According to the view distance, LOD height-field textures can also be retrieved on every height level of the quad-tree. To achieve high accuracy, a large number of height-field textures may be involved in a rendering step.

We only render the tiles which are in the view frustum. In order to visualize shadows on the terrain, we compute a horizon map texture for each visible LOD height-field texture. This enforce the consideration of neighboring tiles. Therefore, all required textures are uploaded to the GPU memory and inserted into a CUDA array which manages all the tiles available in the GPU memory. Thus, it is possible to get access to each height-field texture by means of the CUDA API. The CUDA array index helps to specify neighboring tiles quickly. As a result, all needed tiles are available and accessible on GPU to compute the horizon maps needed for the current rendering loop. To increase the performance even more, the horizon maps are only computed when the position of the light source changes.

Due to the LOD approach, the computation of the horizon maps can be quite complex. To compute the horizon angle, the algorithms has to traverse along titles with different LOD resolutions. According to the speed of the navigation over the terrain, LOD levels have to change and the render performance varies. This mainly depends on the implementation of the LOD algorithm. In this paper, we only demonstrate that the proposed method is compatible with LOD approaches.

5 Results

We have tested our algorithm with different resolutions and databases. For evaluation purpose, we use simple artificial data such as a cube to verify our results (Figure 4). Such rendered shadows can easily be validated against exact shadows calculated by geometrical formulas. However, we also test our results with real data from different celestial bodies such as the Moon (Figure 5). Those datasets are much more complex. To evaluate those results, we change the elevation angle interactively and assess the shadows cast by the mountains and craters visually.

In Table 1, the performance statistics for different height-field resolutions are given. To compare our results, we also implemented the previous method [Max88] with 16 horizon directions (Figure 1). The benchmarks have been executed on a single workstation. This workstation includes a single Nvidia Quadro 6000 with ECC memory (total 6 GB GDDR5 memory) and fast double precision capabilities. Furthermore, it is equipped with one Intel Xeon 2.4 GHz 8 core processor and 24 GBs of DDR3 memory.

We have observed a mean value of 527 fps (Frames Per Second) for a single tile with a resolution of 256x256 as depicted in Figure 5. The conventional horizon mapping method has achieved a mean value of 1542 fps with 0.37 seconds of pre-computation time. Our method does not require any pre-computation. For a 512x512 resolution, we have observed a mean value of 89 fps. With this resolution, the conventional horizon mapping is faster (355 fps).

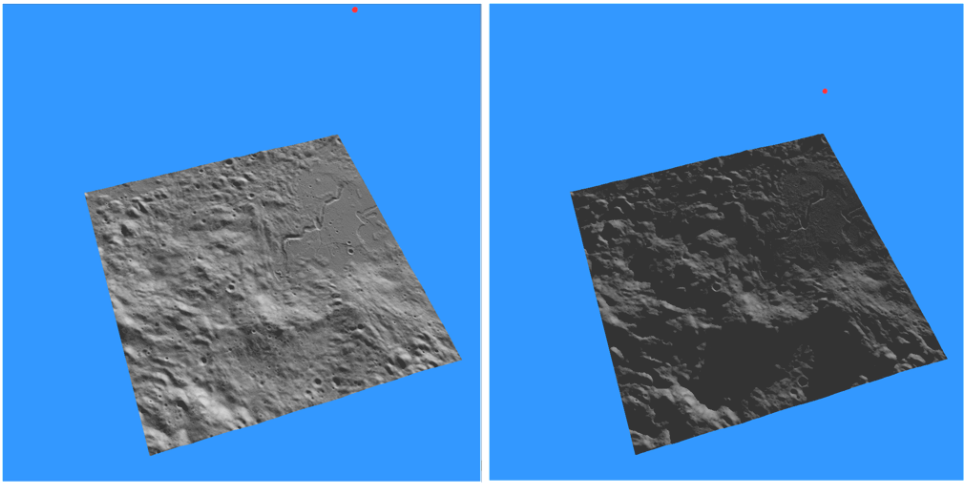


Figure 5: Moon, Apollo 15 landing area, 512x512 grid resolution, Kaguya, SELENE dataset (JAXA). (a) High elevation angle; (b) Low elevation angle.

Table 1: Performance statistics for different height-field resolutions.

Resolution	Proposed Method (fps)	Conventional Horizon Mapping (fps)
256x256	527	1542 (0.37 s Pre-computation Time)
512x512	89	355 (2.9 s Pre-computation Time)
1024x1024	12	154 (21.93 s Pre-computation Time)

But pre-computation time is also increasing (2.9 seconds). Shinoyama and Max [SM10] got 162 fps, 40.0 fps, 39.2 fps with cone mapping, spherical radial basis functions (SRBF), and conventional horizon mapping respectively. They used 32 azimuthal directions in their implementation. We have achieved 12 fps with a 1024x1024 resolution while the conventional horizon mapping implementation has 154 fps with 21.93 seconds for pre-computation. In general, the conventional horizon mapping is faster but requires an increasing pre-computation time.

The GPU memory usage of the proposed method naturally increases along with the resolution of the height-field. In Table 2, the texture memory represents the amount of the memory that is occupied by the horizon map texture. The memory values for conventional horizon mapping are given for the horizon mapping method with 16 azimuthal directions. It is obvious that our memory requirement is less than the conventional horizon mapping method. The original one requires more than 16 horizon map textures to compute visibility more accurately. And its textures have pixel values with floating point precision[1]. In

[1]Memory can be optimized by usage of RGBA textures as Sloan and Cohen demonstrated [SC00]. However, this decreases accuracy since horizon angles are generally not integers but floating point numbers.

contrast, our method only requires a single texture with black and white values.

Table 2: Memory usage (KB) for different height-field resolutions

Resolution	Proposed Method	Conventional Horizon Mapping
256x256	64	1024
512x512	256	4096
1024x1024	1024	16384

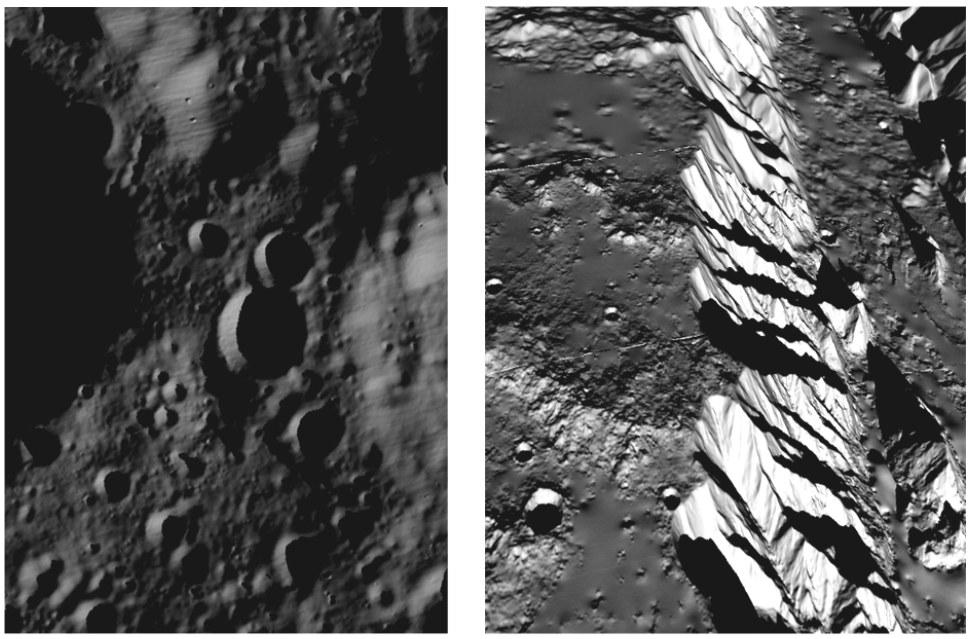
The memory efficiency is not highly demanded to render a single height-field texture. This is simply because modern GPUs have sufficient memory (1 to 6 GB, and more). On the other hand, real world applications use hundreds of textures to render e.g. a complete earth surface. Therefore, the memory should be exploited efficiently. Moreover, the pre-computation of horizon textures requires long computation time. As a result, the proposed method is more suitable for those applications.

As we have mentioned in the previous section, we have also implemented the proposed method with a LOD approach. Our application has to handle approximately 4 TB of an Digital Elevation Model (DEM) database (Figure 6 (a)). We have observed real-time rendering (20 to 30 fps). The visibility maps are computed for a number of tiles varying between 12 and 60. Each tile possesses a 255x255 resolution height-field texture. The number of tiles depends on the camera position and orientation. Horizon map textures together with the height-field textures occupy up to 1 GB GPU memory. To implement our LOD approach with conventional horizon mapping (with 16 azimuth directions), we would need 16 TB of memory for the pre-processed data.

6 Conclusion and Future Work

We have proposed an alternative rendering technique for generating accurate shadows on height-fields. The implementation is based on the horizon mapping algorithm. Our results are compared to the previous methods. We also depicted preliminary results with regard to large terrain datasets. The main contributions of our approach are:

- A real-time method without artifacts.
- Flexible approach, compatibility with LOD data structures.
- Minimal memory usage and efficient handling of large data.
- Simple, straight forward implementation.



(a) (b)

Figure 6: The proposed method is also compatible with the LOD approach. (a) The dataset belongs to the Moon surface, north pole [Arc15]. Hard shadows in the craters and on the valley can be clearly seen. (b) The dataset belongs to the Mars surface (Valles Marineris) [NAS].

Our method also has some limitations. First, it can only be used for self-shadowing of the terrain. But it might be used together with other shadowing techniques to cast shadows of further objects (like Rover models) onto the terrain. Second, the complexity of the algorithm might increase dramatically with multiple light sources. Therefore, the use of the approach in real-time applications might be limited to the height-fields with a single light source such as the Sun.

As future work, our method can be investigated to accelerate the core algorithm and improve its scalability. The compatibility of our method with other LOD approaches should be assessed. The memory requirements and frame rates with respect to resolutions of large terrain datasets can be analyzed in more detail. Furthermore, the performance issues might be evaluated with regard to multiple light sources.

7 Acknowledgements

This work has been supported by DLR internal funding.

References

[Arc15] SELENE Data Archive. Kaguya dataset:http://l2db.selene.darts.isas.jaxa.jp. 2015.

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH Comput. Graph.*, 12(3):286–292, August 1978.

[BM93] Barry G. Becker and Nelson L. Max. Smooth transitions between bump rendering algorithms. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 183–190, New York, NY, USA, 1993. ACM.

[CD03] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGRW '03, pages 208–218, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Lai06] Samuli Laine. *Efficient Physically-Based Shadow Algorithms.* PhD thesis, Helsinki University of Technology, 2006.

[LKR+96] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 109–118, New York, NY, USA, 1996. ACM.

[Max88] Nelson L. Max. Horizon mapping: shadows for bumpmapped surfaces. *The Visual Computer*, 4:109 – 117, 1988.

[NAS] NASA. Lola dataset : http://pds-geosciences.wustl.edu/missions/lro/lola.htm, 2015.

[NS09] Derek Nowrouzezahrai and John Snyder. Fast global illumination on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering*, 2009.

[nvi15] http://www.nvidia.com/, 2015.

[OMN04] K. Onoue, N. Max, and T. Nishita. Real-time rendering of bumpmap shadows taking account of surface curvature. In *Cyberworlds, 2004 International Conference on*, pages 312–318, Nov 2004.

[SC00] Peter-Pike J. Sloan and Michael F. Cohen. Interactive horizon mapping. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 281–286, London, UK, 2000. Springer-Verlag.

[SM10] Noriaki Shinoyama and Nelson Max. Fast height-field rendering under image-based lighting. *Pacific Conference on Computer Graphics and Applications*, 0:24–31, 2010.

[SN08] John Snyder and Derek Nowrouzezahrai. Fast soft self-shadowing on dynamic height fields. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*, EGSR '08, pages 1275–1283, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[Wol11] David Wolff. *OpenGL 4.0 Shading Language Cookbook*. Packt Publishing, 2011.

[WSBW01] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. In *Computer Graphics Forum*, pages 153–164, 2001.

[WWT+03] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. View-dependent displacement mapping. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 334–339, New York, NY, USA, 2003. ACM.