High-Performance Graphics (2025)
A. Knoll and C. Peters (Editors)

Fast Planetary Shadows using Fourier-Compressed Horizon Maps

J. Fritsch[1], S. Schneegans[2], F. Friederichs[3], M. Flatken[1], M. Eisemann[3], A. Gerndt[1,2]

[1]German Aerospace Center, Institute of Software Technology, Germany
[2]University of Bremen, High-Performance Visualization Group, Germany
[3]TU Braunschweig, Computer Graphics Lab, Germany

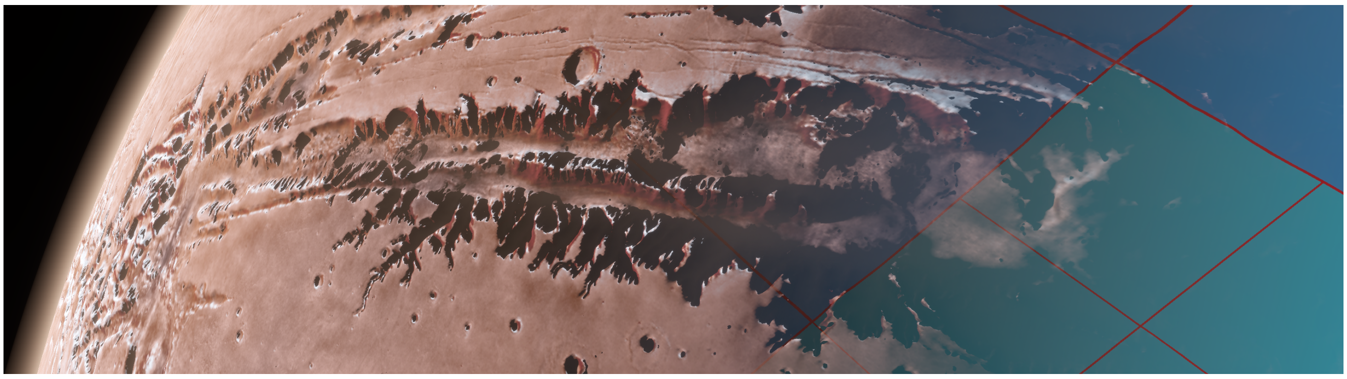


Figure 1: *Rendering of Valles Marineris on Mars utilizing Fourier-compressed horizon mapping inside CosmoScout VR. Our horizon maps are organized in a quad-tree, only loading in-frustum tiles at a resolution suiting their screen space size. Tile borders are visualized on the right side of the figure. The scene can be fully lit in less than 4ms on a notebook equipped with an Nvidia RTX 2050.*

Abstract

Shadows on large-scale terrains are important for many applications, including video games and scientific visualization. Yet real-time rendering of realistic soft shadows at planetary scale is a challenging task. Notably, many shadowing algorithms require keeping significant amounts of extra terrain geometry in memory to account for out-of-frustum occluders. We present Fourier-Compressed Horizon Mapping, an enhancement of the horizon mapping algorithm which is able to circumvent this requirement and render shadows in a single render pass. For a given digital elevation model, we create a compact representation of each pixel's horizon profile and use it to render soft shadows at runtime. This representation is based on a truncated Fourier series stored in a multi-resolution texture pyramid and can be encoded in a single four-channel 32 bit floating point texture. This makes this approach especially suitable for applications using a level-of-detail system for terrain rendering. By using a compact representation in frequency space, compressed horizon mapping consistently creates more accurate shadows compared to traditional horizon maps of the same memory footprint, while still running well at real-time frame rates.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Rendering;***

1. Introduction

Shadows play a crucial role in the visual perception system of humans. They provide our brain with essential information about the three-dimensional structure of objects [MKK98]. This is especially true for large-scale shadows on terrain, where they can help to estimate distances, shapes, and the direction to the Sun. Hence, there are many applications that benefit from the presence of shadows, such as video games, scientific visualization of remote sensing data, or space mission planning. Furthermore, rendering correct shadows is not only required for human perception, but also for machine learning, e.g. for training and testing of optical terrain-relative navigation algorithms [KLBS15].

However, generating high-quality shadows for large-scale terrains is a non-trivial task. Such terrains are often represented as

DOI: 10.2312/hpg.20251171

heightmaps, also known as digital elevation models (DEMs) in geographical contexts and are typically stored in a level-of-detail (LOD) system. In such systems, only the visible terrain tiles should be kept in memory to save resources and maximize the visual quality. Yet terrain features outside the view frustum could still cast shadows on the visible terrain which would require keeping this data in memory as well. Furthermore, at planetary scale, mountains are very small but are able to cast extremely long shadows during sunset and sunrise. In such scenarios, traditional shadow mapping algorithms often struggle with depth-precision issues. Hence, the required parameters heavily depend on the spatial configuration of terrain, observer and light direction.

One solution to this problem is horizon mapping [Max88]. The general idea of horizon mapping is to precompute the profile of the visible horizon for each point of the DEM. During real-time rendering, the Sun elevation is mapped into this space and compared with the horizon elevation. The advantage of this method is that all information required to compute soft shadows for a position on the terrain is available locally and hence it works well with tile based LOD systems. While this method can produce accurate shadows in a very fast manner, a precise horizon shape requires an significant amount of data to be stored per DEM location. For our work, we require that all horizon data can be stored in a single texture usable by OpenGL. This limits us to 4 color channels of 32 bits each, but allows us to use additional smaller resolution mipmap levels.

In this paper, we extend traditional horizon mapping with a compression step, allowing us to satisfy these requirements while still producing satisfying shadows. Our fundamental idea is to use a Fourier transformation of the horizon profile and store the coefficients spread over the different levels of the mipmap pyramid. We evaluate different compression levels, compare the performance and visual quality of our method with the traditional horizon mapping approach, and present different ways to generate soft shadows based on the horizon information. As a proof of concept, this method was implemented into CosmoScout VR, an open-source scientific visualization tool for planetary data [SZGG22].

In the next section, we give a detailed overview on previous work. Then, we introduce the implemented method to produce compressed horizon maps. Thereafter, an in depth evaluation of this method is presented. Lastly, we conclude with a summary and present ideas for future development.

2. Related Work

Generating shadows in graphics applications is a compute and memory intensive task. Varying target applications such as video games, photo-realistic rendering, or the generation of images for optical terrain-relative navigation algorithms have different requirements on accuracy and performance. Because of this, rendering of shadows is a wide field of research. While our work focuses on the generation of real-time soft shadows for large terrains, we suggest reading the book by Eisemann [ESAW11] and the survey on soft shadows by Hasenfratz [HLHS03] for completeness.

While ray tracing can be used to render highly accurate soft shadows [CPC84], it is computationally the most expensive solution. In order to increase the rendering performance, numerous parallel and distributed approaches also exploiting modern graphics cards have been developed [SFG20, MWR12]. However, these ray-tracing solutions require the geometry of all occluders to be in memory at runtime. For large-scale DEMs reaching multiple terabytes in size, this is difficult or nearly impossible.

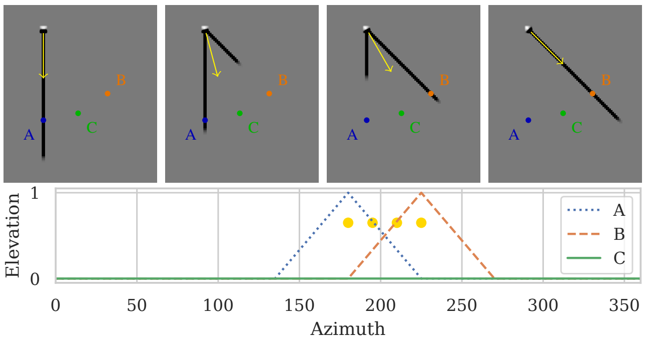


Figure 2: ***Top row:*** *Top-down view of shadows cast by a small peak (white dot in top left) for different solar azimuths (yellow arrow).* ***Bottom row:*** *Local horizon function at points A, B, C, as well as the solar positions mapped to spherical coordinates (yellow dots). Uncompressed horizon mapping can only render shadows along the discrete number of sampled directions. For points A and B, the peak lies on a sampled direction, while the horizon function for point C is completely flat.*

Optimized approaches for such terrains include ray marching algorithms on implicit surfaces defined by the DEMs. These algorithms search for the point of intersection with the implicit surface in an iterative fashion, which requires many expensive accesses to the underlying height map. Optimizations, such as maximum mipmaps, can be applied to skip empty space and reduce the number of texture accesses [TIS08, JSS20]. While these solutions are optimized for 2D domains formed by the DEMs, they still require the full resolution to be available at runtime.

An efficient alternative to solutions based on ray tracing are shadow mapping techniques, where depth renders from the light sources are used to decide if fragments are shadowed [Wil78, Dim07]. Whereas the original approach provides only hard shadows, many adaptations for soft shadows have been developed [RSC87,Fer05,DL06]. However, a mismatch between shadow map texels and screen space fragments can lead to significant artifacts, especially for large scale planetary scenes at shallow illumination angles. Additionally, these methods again require occluding geometry to be in memory at all times.

In order to avoid excessive computational and memory overheads from loading all possible occluder geometry, there exist shadow algorithms which move selected calculations to a preprocessing step. In precomputed radiance transfer, the radiance transfer function is measured at a set of probe points on arbitrary geometry under various incoming light configurations. For each probe, the function is stored as a sum of spherical harmonics basis functions [SKS02]. This transfer function is defined on a 2D domain, while our approach only needs to represent a function on a 1D domain, thus requiring fewer coefficients to achieve a similar angular

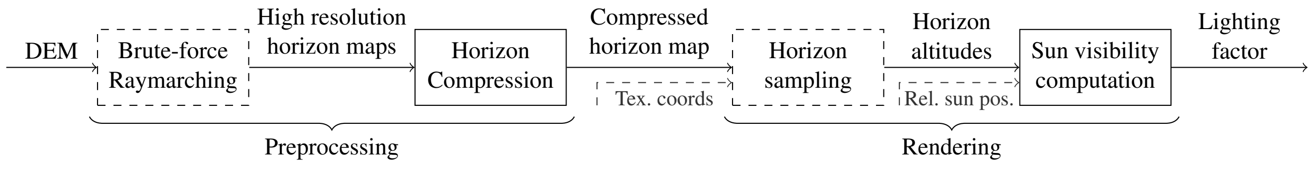


Figure 3: *Overview of our method. While we introduced novel ideas to all steps, our main contributions are marked by a solid border.*

resolution. Additionally, we support high frequency illumination such as point lights and area lights with low angular diameter.

By restricting the geometry to surfaces described by heightmaps, the horizon mapping algorithm can be used [Max88]. Here, for a point $\mathbf{p}$ on the surface M the visibility of light sources is determined in the local tangent space at $\mathbf{p}$. Let the functions $\theta_{\mathbf{p}}(\cdot)$ and $\alpha_{\mathbf{p}}(\cdot)$ compute the azimuth (angle around normal vector) and elevation angles (angle above tangent plane) for another point $\mathbf{p}'$, relative to $\mathbf{p}$. With this, we can define the *horizon function*:

$$h_{\mathbf{p}}(\theta) = \max_{\{\mathbf{p}' \in M | \theta_{\mathbf{p}}(\mathbf{p}')=\theta\}} \alpha_{\mathbf{p}}(\mathbf{p}') \tag{1}$$

To compute visibility for a specific incident light direction ω_i, one simply compares the elevations $\alpha_p(\omega_i)$ and the corresponding value from the horizon function:

$$V(\mathbf{p}, \omega_i) = \begin{cases} 1 & \text{if } \alpha_{\mathbf{p}}(\omega_i) > h_{\mathbf{p}}(\theta_{\mathbf{p}}(\omega_i)) \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The total contribution of the light source is then computed as the integral of the product of incident radiance and visibility over the solid angle subtended by the light source. In traditional horizon mapping, $h_{\mathbf{p}}$ is approximated by a piece-wise linear discretization using eight directional samples, and stored in a set of 2D textures, called *horizon map*, which is computed only once in a preprocessing step. Then, during rendering, the horizon function is reconstructed using linear interpolation between the discrete sample points. While this converges to perfect shadows under direct lighting at high azimuthal resolutions, at the eight samples used in Max' original method aliasing artifacts occur, as the true horizon function is severely under-sampled. Points on the terrain can only receive shadows, if an occluder lies in one of the sampled directions. For sharp and narrow features shadows may only occur on straight lines in the corresponding azimuthal directions, while the space between these is always illuminated. This leads to the multiple shadows shown in fig. 2 appearing when the light comes from a non-sampled direction.

An early implementation of this algorithm using the fixed function graphics pipeline was provided by Sloan et al. [SC00]. There are further works building on ideas from the horizon mapping algorithm. Most conceptually similar to our approach is the work by Heidrich et al. [HDKS00]. Here, the local horizon is represented as an ellipse on the hemisphere surrounding each point. This results in a fixed amount of six degrees of freedom, whereas our chosen representation can support a dynamic number of degrees of freedom depending on the memory available. While several works on executing horizon mapping fully online without the preprocessing step exist [SN08, NS09, TW10, AFG15], these again require geometry to be loaded at runtime.

Apart from being used for determining direct illumination, horizon functions can also be used for computing ambient occlusion [BSD08] or estimating snowmelt in alpine regions caused by incoming solar radiation [Arn87].

3. Compressed Horizon Mapping

The key idea of our method is to create an intermediate horizon representation that is capable of preserving notable features while keeping to a small memory footprint. We especially aim to improve on the main drawback of the traditional method, namely the erasure of features not aligned with the discrete set of azimuthal directions.

Our method can be divided into four steps arranged as a pipeline which is shown in fig. 3. We start by gathering high resolution horizon maps using a naive brute-force raymarching algorithm on an input DEM. These horizon maps are then compressed to a more compact representation fulfilling the previously specified requirements. These first two steps are executed as an offline preprocessing stage and produce compressed horizon map files, which are then loaded by the rendering application at runtime. During the rendering stage, one or multiple values are sampled from the compressed horizon function to either compute soft or hard shadows. As output, we produce a *lighting factor* in the interval $[0,1]$, with 0 assigned to completely shadowed pixels and 1 assigned to completely illuminated pixels. In the simplest case, this is then multiplied with the fragment color to produce the final shader output. In the following subsections, each of these steps is presented in more detail.

3.1. Computing High Resolution Horizon Functions

Our pipeline starts by reading in a DEM from a GeoTIFF file and computes a high resolution horizon function for each pixel of the DEM. We implemented this in a fully parallelizable way where the computations for each pixel and each azimuthal direction are completely independent from each other. While this is sufficient for demonstrating our method, optimizing this algorithm presents an opportunity for future research.

As our method is intended for the rendering of planetary scale DEMs, we have to include the effects of planetary curvature and of the DEM's map projection during raymarching. Instead of marching along straight lines in texture space, we first take a pixel's texture space position P_0 and its height according to the DEM, transform it into a position P_0' in 3D-space on a planetary ellipsoid and then construct sample points $P_{k,i}'$ at regular intervals along K rays r_k. Each ray lies in the plane defined by P_0' and its normal vector N pointing away from the ellipsoid's center. The rays are constructed by starting with the ray in positive latitudinal direction r_0 and rotating it around N by $\frac{k}{K}2\pi$ radian. The sample points $P_{k,i}'$ are then

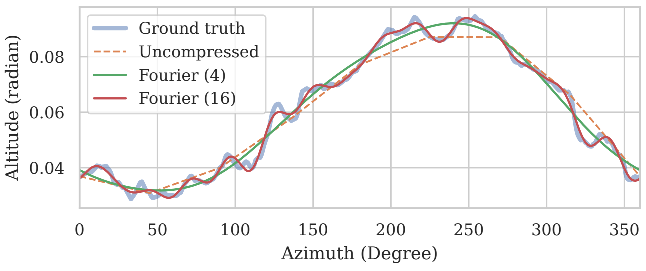


Figure 4: *Truncated Fourier series representation of a horizon (4 and 16 coefficients) compared to ground truth and a traditional horizon map with 8 support points.*

transformed back into texture space to look up their corresponding height value $h_{k,i}$ in the DEM. The coordinate transformations are done using the GDAL library [GDA25].

For calculating the horizon elevation angle at each sample point, we use the triangle formed by the start position of our ray P'_0, the sample point $P'_{k,i}$, and the point on the ellipsoid's surface $P_{k,i}$ directly above or below $P'_{k,i}$, found using the height value $h_{k,i}$ from the DEM. The horizon elevation angle then is $\alpha = \angle P'_{k,i} P'_0 P_{k,i}$ and can be computed using the law of cosines. The maximum elevation angles encountered along each ray form the local horizon function.

This yields a three dimensional array of elevation angles, in which the angle for the k-th azimuth θ_k at pixel p can be accessed at the index $[k, p_x, p_y]$. If the horizon is sampled in 1° steps, the size of the resulting array is about 360 times the size of the input DEM, which is infeasible to use at runtime. Therefore, we compress the data in the following step.

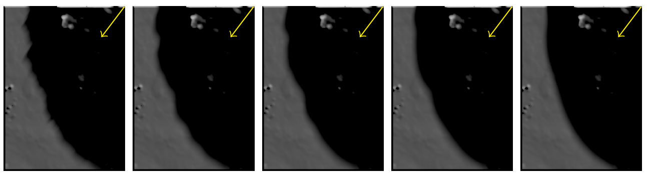
3.2. Compression

The memory footprint of the previously computed horizon map must be greatly reduced to be viable for rendering. To achieve this, traditional horizon mapping [Max88] subsamples the horizon function at eight fixed positions, which are then linearly interpolated during rendering. We implement this approach as our baseline.

To improve on this, we use truncated Fourier series. First, we compute the discrete Fourier transform (DFT) of each pixel's horizon function to get its representation in frequency space and then we remove Fourier coefficients corresponding to high frequencies until we meet our desired memory requirements. An example of the resulting horizon shape can be found in fig. 4.

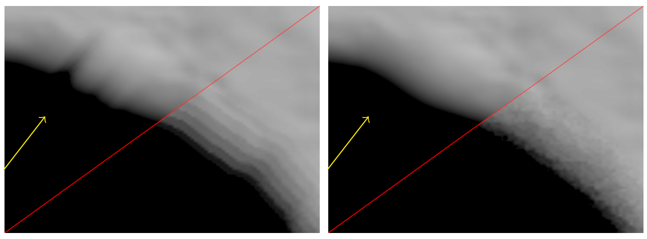
While this method inherently applies a low-pass filter to the horizon function, it provides several key advantages: When thinking of each Fourier coefficient as a pair of phase and amplitude values, it becomes clear that this representation allows arbitrary placement of low-frequency features on the horizon by gradually shifting the phase value while keeping the amplitude constant. Additionally, interpolation between two complex numbers in vector form produces sensible intermediate values. Also, the inherent periodicity of Fourier series guarantees a continuous and periodic horizon.

To further tune this method, we explored multiple parameters that influence both visual quality and degree of compression.



(a) *GT* **(b)** *16 Coeffs.* **(c)** *12 Coeffs.* **(d)** *8 Coeffs.* **(e)** *4 Coeffs.*

Figure 5: *Soft shadows cast by a cliffside, incoming light direction shown by yellow arrow, rendered using high resolution horizon map (*GT*) and Fourier-compressed horizon maps with varying amounts of coefficients evaluated (*n Coeffs.*). The number of Fourier coefficients directly influences the accuracy of shadows. While subfigure (e) depicts a smoothed shadow contour, subfigure (b) produces more accurate results.*



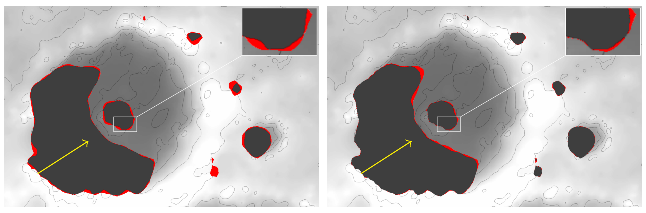
(a) *Traditional horizon map* **(b)** *Compressed horizon map*

Figure 6: *Soft shadows cast by a cliffside, incoming light shown by yellow arrow, rendered with 16-bit (top left) and 8-bit (bottom right) horizon maps. The decrease in bit depth results in banding artifacts for traditional horizon maps and noisy penumbras for compressed ones. Adjusted white point to highlight artifacts.*

Number of coefficients: The primary parameter for adjusting the balance between visual quality and compression degree is the number of Fourier coefficients. As shown in fig. 5, the known low-pass filter effect of truncated Fourier series is also exhibited by the contour of the global shadow. However, the effect is gradual and there is no clear cut-off where the shadows seem to be completely implausible. Therefore, this parameter is left open and is suitable as a runtime parameter for adjusting the performance of our method.

Bit depth: We also explored storing the Fourier coefficients at different bit depths. To clarify, the bit depth b given here always concerns a single component of one complex Fourier coefficient. I.e., to store n complex coefficients, $2nb$ bits of memory are required.

We observed no discernible difference between shadows rendered with 32-bit floating point numbers (*singles*) and 16-bit floating point numbers (*halfs*). To go from 16-bit halfs to 8-bit unsigned integers, we normalized all coefficients and stored minimum and maximum values per coefficient and horizon map as metadata. Storing separate values for each coefficient is necessary, as there is a difference of two orders of magnitude between the maxima of low frequency terms (~ 0.23) and high frequency terms (~ 0.0015). As depicted in fig. 6, this leads to noticeable artifacts for both our Fourier-compressed horizon maps as well as for traditional horizon mapping even for local DEMs with limited value ranges. Thus, we use 16-bit halfs for both methods during our evaluation.



(a) *Global downsampling* (2×2) **(b)** *Coefficient* k_i *downsampled by* $2^{\lfloor \frac{i}{4} \rfloor} \times 2^{\lfloor \frac{i}{4} \rfloor}$

Figure 7: *Comparison of hard shadows rendered using downsampled horizon maps to reference images rendered without downsampling (all Fourier-compressed with 16 coefficients). Solid gray areas mark true positive shadows/pixels, red areas mark false negative or false positive shadows/pixels. Scene shows a crater with incident light indicated by yellow arrow.*

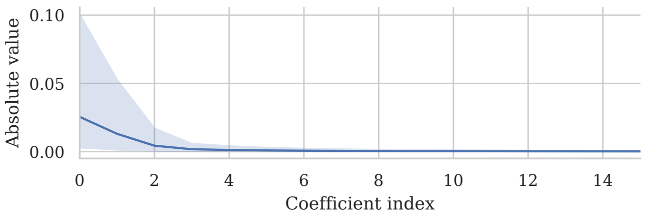


Figure 8: *Mean absolute amplitude and corresponding 95% percentile for each coefficient. Lower indices correspond to lower frequency terms.*

Spatial resolution: The required memory footprint to store the horizon maps can be further decreased by reducing their spatial resolution. This is done by applying a 2×2 box filter to them in the x-y-plane, repeatedly until the desired resolution is reached. This spatial subsampling produces noticeable artifacts at the base and tip of the shadows as depicted in fig. 7a. While this might be acceptable at the tip of the shadows, the changes at the base can lead to shadow or light bleeding near the edge of an occluder.

Since the global sampling method leads to unsatisfying results, especially at the base of each shadow, we further utilize selective downsampling. Here, more accurate shadows can be preserved by only reducing the spatial resolution of high frequency coefficients. As depicted in fig. 8, the mean absolute value of each Fourier coefficient decreases with rising frequency. This means that the low frequency terms have the strongest impact on the resulting horizon shape. By keeping the corresponding low frequency coefficients at a high spatial resolution, but downsampling higher frequency coefficients with filters of increasing size, the resulting image in fig. 7b depicts only slight changes in the base of a shadow. The increased error in the tip of the shadows comes from the larger filter size for the highest frequency coefficients, e.g. coefficients 12-16 are downsampled with a 8×8 filter in fig. 7b, while in fig. 7a only a 2×2 filter is applied to them. However, we see this error as less disruptive compared to the errors occuring across all shadow edges.

In order to efficiently store such multi-resolution horizon maps with a spatially decreased resolution for different Fourier coefficients, we exploit mipmap pyramids. For our selected bit depth of 16-bit, we can store up to 4 coefficients per level using a 4-channel, 32-bit per channel texture. Specifically, this results in the first four coefficients being stored at full spatial resolution $r_x \times r_y$, the next four to be stored at $\frac{r_x}{2} \times \frac{r_y}{2}$, etc. We observed diminishing returns after a minification factor of 16, so we limited the pyramid to four levels with 16 coefficients in total.

Summary: In conclusion, our method truncates the Fourier series to 16 coefficients and stores these as pairs of 16-bit floating point numbers. These are grouped into blocks of four coefficients each, with the first block containing the coefficients of the lowest frequency basis functions. The first block is stored at native resolution, each further block is stored at half the resolution of the previous one. For persistent storage of these compressed horizon maps, we use TIFF files as these support arbitrary amounts of image layers with varying resolutions. Our implementation stores each 32-bit complex coefficient in two separate 16-bit floating point layers on disk, resulting in eight total layers per resolution level.

3.3. Rendering

At runtime, the compressed horizon maps are read from disk and uploaded to the GPU as a single additional 4-channel texture layer. To do so, we store two of the previously mentioned 16-bit floating point layers in a single color channel in the high and low bytes of a 32-bit integer channel. This results in four complex Fourier coefficients being stored in each mipmap level of the texture.

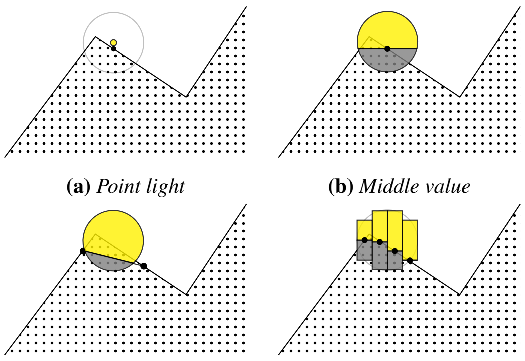
This texture is then accessed in the shader for terrain rendering. As Fourier coefficients can be interpolated, we can also access non-pixel-aligned texture coordinates. Note though that this has to be implemented in the shader code as our packing scheme is not supported by hardware interpolation. Due to possible interpolation, either the vertex or fragment shader can be used for computing shadows. This decision has both performance as well as quality implications: Assuming that the triangles of the terrain have a size of multiple fragments in screen space, the fragment shader is invoked more often than the vertex shader. On the other hand, assuming the horizon map has a higher resolution than the DEM, using the fragment shader can show additional details. Apart from this, the operations carried out in both shaders are the same. In the following, *point* refers to either a vertex or fragment.

In order to compute shadows for a given point P, the light source is projected onto the unit sphere surrounding P and described using spherical coordinates. For a point light, this naturally results in a single pair of spherical coordinates. Then, only a simple binary check is required to light P: If the elevation of the light source α_s is larger than the horizon elevation α_h in the azimuthal direction of the light source θ_s, P is illuminated, otherwise P stays dark.

To compute α_h at P, we add up the N basis functions weighted by the Fourier coefficients $\{X_k = X_k^r + X_k^i \cdot i, k \in [0, N-1]\}$:

$$\alpha_h = \sum_{k=0}^{N-1} (X_k^r \cdot \cos(k\theta_s) - X_k^i \cdot \sin(k\theta_s)) \tag{3}$$

We assume sine and cosine functions to be reasonably optimized by GPU vendors, so we use plain `sin` and `cos` GLSL functions. As shown in section 4.3 this results in adequate performance.



(a) *Point light* **(b)** *Middle value* **(c)** *Circular segment* **(d)** *Rectangular boxes*

Figure 9: *Comparison of different visibility approximations. Black dots mark sampled horizon values.*

While the previous method is valid for producing hard shadows, our intended use-case - illuminating planetary surfaces by the sun's light - requires modeling a spherical light source casting soft shadows. Projecting the sun into spherical coordinates then produces a disk, which we assume emits light uniformly. This ignores the limb darkening effect [HM98] to reduce computational load. Then, the total light arriving at P is proportional to the ratio between the area above the horizon line and the total area of the disk. As the former area is non-trivial to compute analytically, we use the numerical approximations depicted in fig. 9. The equations shown in the following do exactly compute the grey fractions of the solar disk shown in the figure.

Middle value: In the traditional horizon mapping algorithm, a single horizon elevation sampled at the mid azimuth of the sun is used. It is assumed that the elevation is constant over the width of the sun [Max88]. Under this assumption, Max computes the occlusion O for a sampled horizon value α_h, a solar elevation α_s and a solar angular radius r as:

$$h = \text{clamp}\left(\frac{\alpha_h - \alpha_s}{r}, -1, 1\right) \tag{4}$$

$$O = 0.5 + \frac{\sin^{-1}(h) + h \cdot \sqrt{1-h^2}}{\pi} \tag{5}$$

However, this can lead to inaccuracies at the edges of steep flanks in the horizon function. In order to improve the accuracy of the approximation, the next approximations sample additional horizon values around the solar position.

Circular segment: This method takes two samples at $\theta_s \pm r$ and calculates the area of the circular segment above the line through both samples. This can achieve good accuracy for edges with constant slope, but smooths out any features interior to the solar disk as seen in fig. 9c. To calculate this area, we use the points $C = (\theta_s, \alpha_s)$, $L = (\theta_s - r, h(\theta_s - r))$ and $R = (\theta_s + r, h(\theta_s + r))$. First, we project the vector $\vec{LC}$ onto the secant $\vec{LR}$ to get the point C'. The distance between the circle's center C and C' is the apothem d of the circular segment. Then, the area of the segment A_{cs} can be calculated as:

$$A_{cs} = r^2 \cdot \cos^{-1}\left(1 - \frac{r-d}{r}\right) - d \cdot \sqrt{r^2 - d^2} \tag{6}$$

As $A_{cs} <= \pi r^2$ for all cases, we aditionally have to determine if this is the visible or the occluded part of the sun. The occlusion O for P then is:

$$O = \begin{cases} \frac{A_{cs}}{\pi r^2} & C'_y \le C_y \\ 1 - \frac{A_{cs}}{\pi r^2} & C'_y \ge C_y \end{cases} \tag{7}$$

Rectangular boxes: By including even more horizon samples, we can approximate the circle with a series of n regularly spread rectangular boxes of width $\frac{2r}{n}$ and heights corresponding to a sample taken at their centers. This also captures interior features as seen in fig. 9d. The amount of rectangular boxes n can be increased to be able to capture more granular details of the horizon function with increased computational costs. With $n \to \infty$, this converges to the integral of the intersection area. For n boxes, we determine the center positions of each box x_i and their height h_i and then compute the solar occlusion O as:

$$x_i = \frac{r}{n} \cdot (2i+1) - r \tag{8}$$

$$h_i = 2 \cdot \sqrt{r^2 - x_i^2} \tag{9}$$

$$O = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\alpha_i - \alpha_s}{h_i} + 0.5 \tag{10}$$

From these occlusion values, a visibility $V = 1 - O$ can be derived, which can then be used in an existing render engine as a multiplier for the incoming light intensity in order to compute shadows.

4. Evaluation

We implemented our method both in a standalone 2D-renderer and in CosmoScout VR, an open-source scientific visualization tool for planetary data [SZGG22]. The former was used to evaluate the visual quality of the generated shadows, while the latter was used as a real world example for performance measurements. We provide a video showcasing dynamic behavior for both applications in the supplementary material.

4.1. Test Dataset

For evaluating our method, we processed the MOLA DEM of the complete martian surface [FHL18]. The input data has a resolution of 200 meters per pixel, is roughly 10GB in size, and has a resolution of about $100,000 \times 50,000$ pixels. However, due to the high computation cost of our preprocessing, we lowered the resolution to 800 meters per pixel globally. Additionally, we processed selected regions at the full 200 meters per pixel for use in our demonstration video (see supplementary material). We have stored the global horizon map as a set of 3,072 texture patches, each with a resolution of 256×256 pixels. Using our naïve sampling method, each of these patches took around 3 minutes to compute, resulting in 150 compute hours for the complete set. For usage in the LOD system of CosmoScout VR, we additionally created a quadtree with 5 levels by merging and downsampling 2×2 groups of patches. The lowest resolution level in this tree consists of 12 base patches. Planetary surfaces are then rendered as a set of slightly overlapping patches. To reduce aliasing artifacts, these are selected from the quadtree so that each patch fills a similar area in screen space [SZGG22].

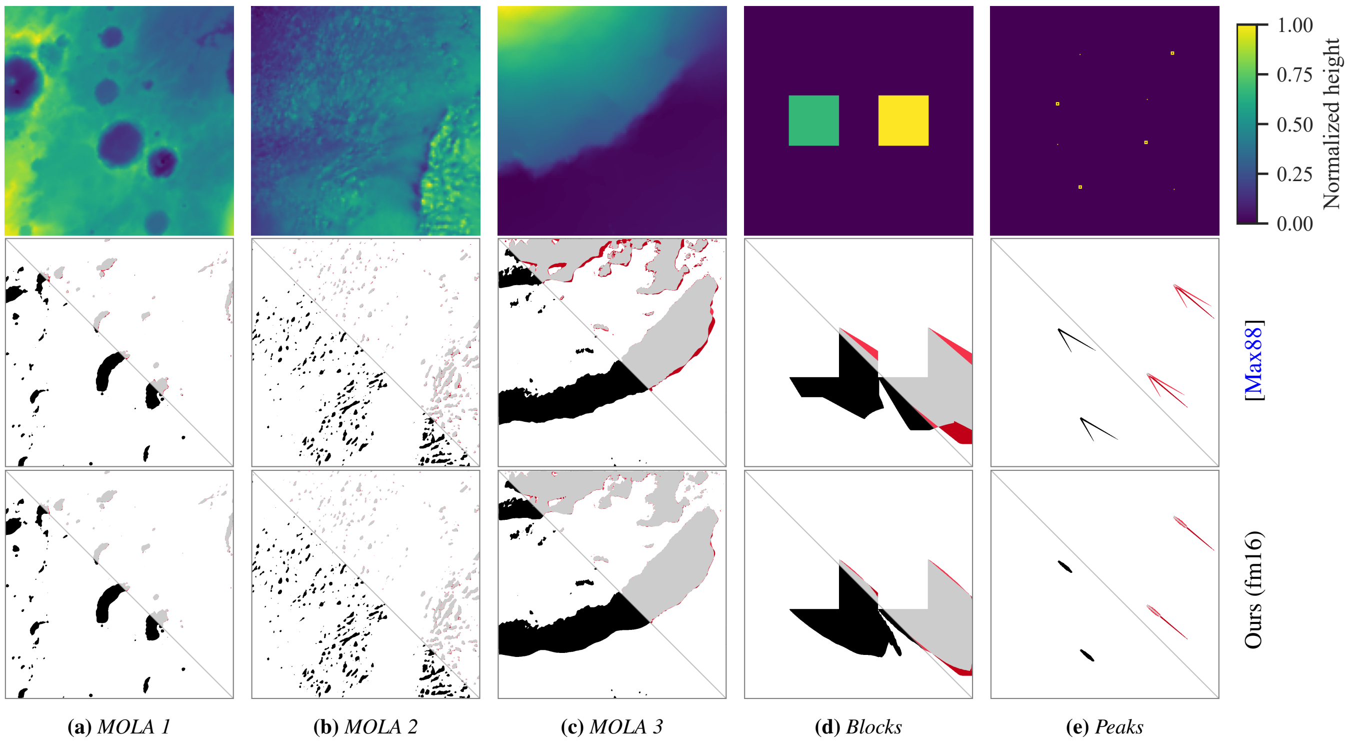


Figure 10: ***1st row:*** *Test DEMs used in the evaluation.* ***2nd row:*** *Hard shadows produced using traditional horizon mapping [Max88].* ***3rd row:*** *Our method (16 Fourier coefficients, stored in 4 mipmap levels). Shadow images show raw shadow data in the lower left corner and a comparison to ground truth data in the upper right corner: Light gray areas mark true positive shadows, red areas mark false negative or false positive shadows.*

During the evaluation, we used multiple different types of horizon maps in addition to our proposed Fourier-compressed horizon maps. An overview of the different types and their file sizes per patch can be found in table 1. Note, that we use twelve samples for the traditional horizon maps instead of the typically used eight samples to ensure a fair comparison with similar memory footprints. The non-mipmapped Fourier-compressed horizon maps are referenced to illustrate the loss in accuracy due to the spatial downsampling. Note however, that these require significantly more memory and do not compete with the other shown methods.

4.2. Visual Quality

For the measurement of visual quality, we have used five test scenes shown in fig. 10, each with different terrain characteristics. We have selected three of the MOLA DEM horizon map patches (*MOLA 1-3*) and additionally have created synthetic worst-case scenes containing small and large features with 90 degree edges, resulting in hard to compress horizons (*Blocks* and *Peaks*).

Our visual quality measurements are based on the comparison of binary hard shadow images, i.e. without computing any additional shading such as the diffuse term. These sets of shadow images have been rendered with multiple settings:

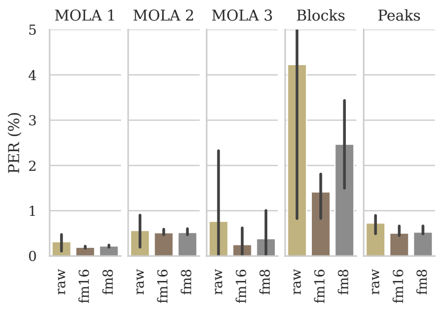
- **Ground truth**: Rendered using traditional, high resolution (360 samples) horizon maps

Texture type	Parameters	Size
High resolution	360 Samples	45MB
Traditional (*raw*)	12 Samples	1.5MB
Fourier mipmap (*fm16*)	16 Coeff., 4 Levels	1.33MB
Fourier non-mipmap (*f16*)	16 Coeff.	4MB

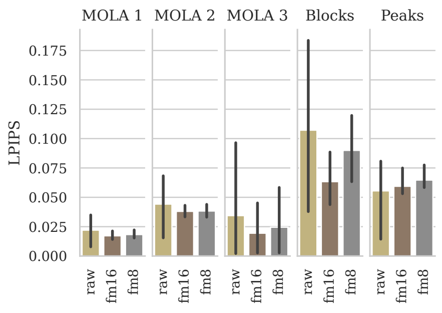
Table 1: *File sizes per terrain patch used during evaluation.*

- **Traditional (*raw*)**: Rendered using traditional, low resolution (12 samples) horizon maps
- **Mipmapped Fourier (*fm4,8,12,16*)**: Rendered using Fourier-compressed horizon maps with 4, 8, 12 and 16 coefficients, with spatial resolution decreasing every four coefficients
- **Non-mipmapped Fourier (*f8,12,16*)**: Rendered using Fourier-compressed horizon maps with 8, 12 and 16 coefficients, with full spatial resolution for each coefficient

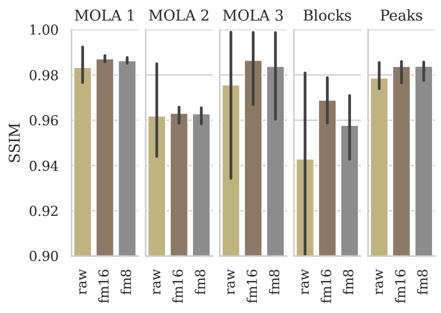
For each of these settings, we have rendered images for sun positions at a fixed elevation angle of 6° and 180 equally spaced azimuths. The visual quality has been measured by pairwise comparisons of the ground truth images and each of the images produced by the additional settings. For each pair, we have determined the ratio between incorrectly shaded pixels and total pixels (*pixel error rate* (PER)) and we have computed both the structural similarity index measure (SSIM) [WBSS04] and the perceptual image similarity metric LPIPS [ZIE*18]. The results are illustrated in fig. 11.



(a) *Pixel error rate. Lower is better.* Blocks-raw *errorbar is cut off, extends to 8.5%.*



(b) *LPIPS. Lower is better.*



(c) *SSIM. Higher is better. Y axis starts at 0.9.*

Figure 11: *Metrics of visual quality for all test scenes, comparing our method at 8 and 16 coefficients to traditional horizon mapping. For each metric and scene, the average value across all captured azimuths is displayed. Errorbars show 80% percentile across azimuths.*

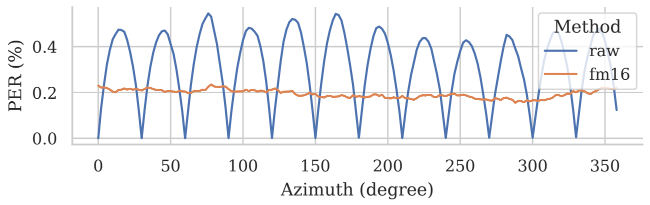


Figure 12: *Pixel error rate as a function of solar azimuth for the MOLA 1 scene.*

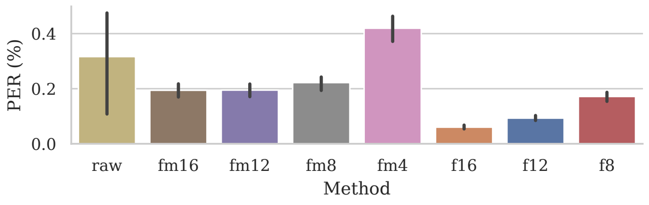


Figure 13: *Pixel error rate for the MOLA 1 scene for all tested method parameterizations. Errorbars show 80% percentile.*

For all but one scene and metric, our method achieves better mean scores (averaged over solar azimuths) both with 8 and 16 coefficients used. The only exception is the LPIPS score for the *peaks* scene. As this metric is determined using a neural network, it is naturally difficult to give a definite reason for this. We suspect, in this case, LPIPS heavily punishes the difference in shape between the sharp lines seen in both the ground truth as well as the *raw* approach and the teardrop shapes rendered by our approach. However, this heavy distortion only occurs in highly artificial scenes. In addition to better mean values, our method also produces more consistent results. As shown in fig. 12, the *raw* approach fluctuates between perfect results for solar azimuths aligned with the twelve sampled horizon elevations and more inaccurate results whenever interpolation is used to compute the horizon elevation. In contrast to this, our method results in similar errors for all azimuths.

When comparing the results for the different scenes, we can see that the improvement, that compressed horizon maps provide over traditional ones, depends on the characteristics of the scene: Scenes with large, coherent shadows such as *MOLA 3* and *Blocks* show significant improvements, while this effect is less pronounced for scenes with smaller, noisy shadows such as *MOLA 2*. Additionally, we can see that the value of using additional Fourier coefficients also follows the same pattern: High frequency terms are mainly relevant when large-scale shadows are present.

Selected results for the additional settings can be found in fig. 13. Here, it becomes apparent that four coefficients are insufficient to match the accuracy of the traditional approach. However, by adding four additional coefficients (i.e. by going from *fm4* to *fm8*), there is a large jump in accuracy. By comparing *fm8* to *f8*, we can see that, while some accuracy is lost from the spatial downsampling of the additional coefficients, the loss is rather small. Further added coefficients provide diminishing returns compared to their non-downsampled counterparts.

Additionally, we compared our proposed ways to produce soft shadows to the approach used by Max [Max88]. Here, we increased the solar radius to 20°, and lowered the solar elevation to 35° to increase the size of the penumbra. Then, we rendered images for 180 equally spaced azimuths again. As ground truth, we used the approximation of the solar disk as a series of 20 rectangular boxes. The resulting mean square errors for each evaluated method can be found in fig. 14. It becomes evident, that using a circular segment (*cs*) or up to three rectangular boxes (*r2*, *r3*) does not improve the quality significantly when compared to using the middle value (*mv*). However, increasing the number of samples further, can produce more accurate soft shadows.

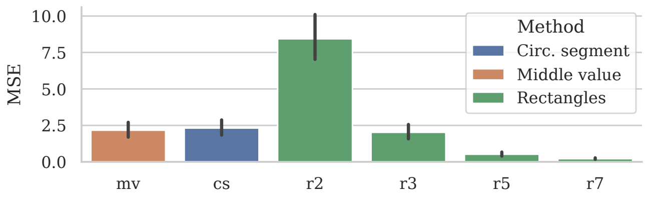


Figure 14: *Mean squared error of different soft shadow approximation techniques based on a Fourier-compressed horizon map. Ground truth is soft shadows with disk approximated as series of 20 rectangles. Errorbars show 80% percentile.*

	Notebook	Desktop
CPU	Intel Core Ultra 7 165H	AMD Ryzen 9 7950X
RAM	64GB DDR5	64GB DDR5
GPU	NVIDIA RTX 2050	NVIDIA RTX 4090
VRAM	4GB	24GB
Driver	Version 538.08	Version 550.67

Table 2: *Specifications of the benchmark test systems*

	Parameters			Frame times (ms)	
Preset	# C.	# S.	Shader	Notebook	Desktop
None	-	-	-	1.21	0.107
Low	8	1	Vert.	2.45	0.204
Medium	16	1	Vert.	4.07	0.297
Medium Frag	8	1	Frag.	5.48	0.363
High	16	1	Frag.	9.98	0.659
Very High x3	16	3	Frag.	14.7	0.535
Very High x5	16	5	Frag.	18.6	0.684
Very High x7	16	7	Frag.	21.2	0.823

Table 3: *Parameters for our proposed quality presets and their resulting average frame times.*

4.3. Performance

We demonstrate our method's capability of rendering shadows in real-time using the implementation in CosmoScout VR. We measure the time per frame taken on rendering a planetary surface using GPU timers. To show the usability both on low- to medium-powered and high-powered hardware, we run these tests on the two test systems described in table 2.

Our test scene consists of a static view of the martian surface. We render the scene at a window resolution of 1920x1080 pixels. Each measurement runs for 30 seconds. The solar radius is set to 0.35°, which is the mean angular size of the sun as seen from Mars.

Based on the following parameters, we propose several quality presets listed in table 3:

- **# C.**: Number of coefficients used at runtime.
- **# S.**: Number of samples used to compute soft shadows. With one sample, we use the middle value approximation. Otherwise, we use the approximation based on a series of rectangular boxes.
- **Shader**: Shader stage that runs the shadow computation.

The measured frame times also include some base cost of rendering the planet, so in addition to these presets, we also measure times without using horizon mapping at all (*None*).

As shown in table 3, on the high-end desktop system, all configurations run in well under 1ms and are easily real-time capable. On the notebook system, the *medium* preset at about 4.1ms (2.8ms overhead over no horizon mapping) is best suited when targetting framerates of 60FPS and more. For the test scene, the selection of the shader stage has the largest effect on performance. However, adequate framerates of about 5.4ms can be achieved in the fragment shader by limiting the number of used coefficients to eight. The different soft shadow computation methods naturally scale with the number of samples being used, as each sample requires to compute a partial inverse DFT for a different azimuth. Still, the coefficients only have to be read once from the horizon map, resulting in a sub-linear scaling behavior. Since the *middle value* soft shadow method proved to provide good visual quality while also being the cheapest to compute, this should be used by default. Then, when performance is of little concern, the rectangular approximation with a higher sample count may be used to further increase the accuracy.

5. Conclusion and Future Work

We presented Fourier-compressed horizon maps, an extension to the horizon mapping algorithm coined by Max [Max88]. Instead of naïvely under-sampling the horizon function, which causes artifacts due to aliasing, our algorithm compresses horizon functions using truncated Fourier series. They are stored in a four-channel, 32-bit per channel texture. Employing three additional mipmap levels for storing higher frequency coefficients at a lower spatial resolution further increases the shadow quality while keeping the memory requirements low. Finally, the horizon map allows for efficient computation of approximate sun visibility and therefore render realistic soft shadows in real-time.

Using our compact horizon representation, we are able to consistently render more accurate shadows compared to the traditional horizon mapping approach while using the same memory footprint per texture. To showcase that our algorithm works in a real-world setup and application, we have integrated the approach into the scientific visualization tool CosmoScout VR. We have demonstrated that our method is capable to render hard and soft shadows at planetary scales with real-time framerates even on low to medium powered notebook graphics cards. We could also show that this solution neatly integrates with multi-resolution LOD systems.

In the future, we intend to use our system for various DEMs with resolutions down to meter-scale. However, the brute-force ray-marching algorithm currently used during preprocessing would require an unreasonable amount of compute hours to produce a complete horizon map for such data. Because of this, we want to look into more efficient ways of producing the initial high resolution horizon maps such as the work done by Stewart [Ste98, TRa11].

Furthermore, while our Fourier-compressed horizon maps show promising results, other compression techniques could be investigated. Especially compression using neural networks has shown to work well recently [LJLB21].

References

[AFG15] Aslandere T., Flatken M., Gerndt A.: A Real-Time Physically Based Algorithm for Hard Shadows on Dynamic Height-Fields. In *Virtuelle und Erweiterte Realität* (Bonn, Sept. 2015), Aachen Verlag, pp. 101–112. 3

[Arn87] Arnfield A. J.: A Fourier Series Approach to Skyline Generalization for Surface Irradiance Estimates in Alpine Terrain. *Arctic and Alpine Research 19*, 3 (Aug. 1987), 270–278. 3

[BSD08] Bavoil L., Sainz M., Dimitrov R.: Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks* (New York, NY, USA, 2008), SIGGRAPH '08, Association for Computing Machinery. 3

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, Jan. 1984), SIGGRAPH '84, Association for Computing Machinery, pp. 137–145. 2

[Dim07] DIMITROV R.: Cascaded shadow maps. *Developer Documentation, NVIDIA Corp* (2007). 2

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, Mar. 2006), I3D '06, Association for Computing Machinery, pp. 161–165. 2

[ESAW11] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-Time Shadows*, 1st ed. A. K. Peters, Ltd., USA, June 2011. 2

[Fer05] FERNANDO R.: Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches* (New York, NY, USA, July 2005), SIGGRAPH '05, Association for Computing Machinery, pp. 35–es. 2

[FHL18] FERGASON R. L., HARE T. M., LAURA J.: *HRSC and MOLA Blended Digital Elevation Model at 200m v2*. Astrogeology PDS Annex, U. S, 2018. 6

[GDA25] GDAL/OGR CONTRIBUTORS: *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation, 2025. 4

[HDKS00] HEIDRICH W., DAUBERT K., KAUTZ J., SEIDEL H.-P.: Illuminating micro geometry based on precomputed visibility. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (USA, July 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 455–464. 3

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F. X.: A survey of real-time soft shadows algorithms. In *Computer Graphics Forum* (2003), vol. 22, pp. 753–774. 2

[HM98] HESTROFFER D., MAGNAN C.: Wavelength dependency of the solar limb darkening. *Astronomy and Astrophysics 333* (1998), 338–342. 6

[JSS20] JUNG D., SCHREMPP F., SON S.: *Optimally Fast Soft Shadows on Curved Terrain with Dynamic Programming and Maximum Mipmaps*. Tech. rep., May 2020. arXiv:2005.06671 [astro-ph] type: article. 2

[KLBS15] KAUFMANN H., LINGENAUBER M., BODENMUELLER T., SUPPA M.: Shadow-based matching for precise and robust absolute self-localization during lunar landings. In *2015 IEEE Aerospace Conference* (2015), pp. 1–13. 1

[LJLB21] LU Y., JIANG K., LEVINE J. A., BERGER M.: Compressive neural representations of volumetric scalar fields. *Computer Graphics Forum 40*, 3 (2021), 135–146. 9

[Max88] MAX N. L.: Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer 4*, 2 (Mar. 1988), 109–117. 2, 3, 4, 6, 7, 8, 9

[MKK98] MAMASSIAN P., KNILL D. C., KERSTEN D.: The perception of cast shadows. *Trends in Cognitive Sciences 2*, 8 (Aug. 1998), 288–295. 1

[MWR12] MEHTA S. U., WANG B., RAMAMOORTHI R.: Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 163:1–163:10. 2

[NS09] NOWROUZEZAHRAI D., SNYDER J.: Fast Global Illumination on Dynamic Height Fields. *Computer Graphics Forum 28*, 4 (2009), 1131–1139. 3

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, Aug. 1987), SIGGRAPH '87, Association for Computing Machinery, pp. 283–291. 2

[SC00] SLOAN P.-P. J., COHEN M. F.: Interactive Horizon Mapping. In *Rendering Techniques 2000* (Vienna, 2000), Péroche B., Rushmeier H., (Eds.), Eurographics, Springer, pp. 281–286. 3

[SFG20] SANZHAROV V. V., FROLOV V. A., GALAKTIONOV V. A.: Survey of Nvidia RTX Technology. *Programming and Computer Software 46*, 4 (July 2020), 297–304. 2

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics 21*, 3 (July 2002), 527–536. 2

[SN08] SNYDER J., NOWROUZEZAHRAI D.: Fast Soft Self-Shadowing on Dynamic Height Fields. *Computer Graphics Forum 27*, 4 (2008), 1275–1283. 3

[Ste98] STEWART A.: Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan. 1998), 82–93. 9

[SZGG22] SCHNEEGANS S., ZEUMER M., GILG J., GERNDT A.: CosmoScout VR: A Modular 3D Solar System Based on SPICE. In *2022 IEEE Aerospace Conference (AERO)* (Mar. 2022), pp. 1–13. ISSN: 1095-323X. 2, 6

[TIS08] TEVS A., IHRKE I., SEIDEL H.-P.: Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games* (New York, NY, USA, Feb. 2008), I3D '08, Association for Computing Machinery, pp. 183–190. 2

[TRa11] TABIK S., ROMERO L. F., AND E. L. Z.: High-performance three-horizon composition algorithm for large-scale terrains. *International Journal of Geographical Information Science 25*, 4 (2011), 541–555. 9

[TW10] TIMONEN V., WESTERHOLM J.: Scalable Height Field Self-Shadowing. *Computer Graphics Forum 29*, 2 (2010), 723–731. 3

[WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (Apr. 2004), 600–612. 7

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, Aug. 1978), SIGGRAPH '78, Association for Computing Machinery, pp. 270–274. 2

[ZIE*18] ZHANG R., ISOLA P., EFROS A. A., SHECHTMAN E., WANG O.: The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 586–595. 7