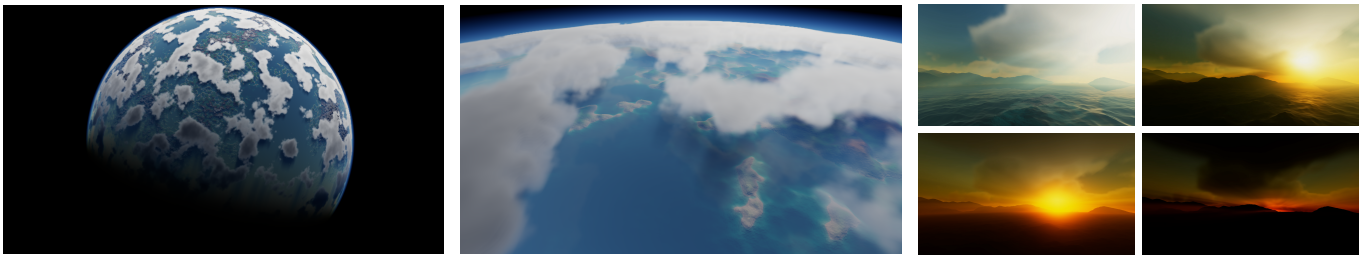
Real-time Rendering of Procedural Planets at Arbitrary Altitudes

FLORIAN MICHELIC and MICHAEL KENZEL, Graz University of Technology, Austria
KARL HAUBENWALLNER, VRVis Research Center, Austria
BERNHARD KERBL and MARKUS STEINBERGER, Graz University of Technology, Austria



(a) Full view of planet, 142 fps (b) Close-up of planet atmosphere, 138 fps (c) Atmospheric effects as seen from surface, 170 fps

Fig. 1. Our holistic method for rendering realistic planets in real-time includes procedurally generated terrain, ocean waves, and clouds in combination with atmospheric scattering. The produced visuals are consistent across arbitrary changes in camera position and proximity: (a) Planet viewed from space, casting a shadow on the surrounding clouds and vice versa; (b) planet viewed from slightly above the atmosphere, with mountaintops reaching out of the haze; (c) multiple snapshots close to the surface during a sunset. Note the clearly visible light shafts. All scenes were rendered and timed using an Nvidia GTX 1080 Ti.

Focusing on real-time, high-fidelity rendering, we present a novel approach for combined consideration of four major phenomena that define the visual representation of entire planets: We present a simple and fast solution for a distortion-free generation of 3D planetary terrain, spherical ocean waves and efficient rendering of volumetric clouds along with atmospheric scattering. Our approach to terrain and ocean mesh generation relies on a projected, persistent grid that can instantaneously and smoothly adapt to fast-changing viewpoints. For generating planetary ocean surfaces, we present a wave function that creates seamless, evenly spaced waves across the entire planet without causing unsightly artifacts. We further show how to render volumetric clouds in combination with precomputed atmospheric scattering and account for their contribution to light transport above ground. Our method provides mathematically consistent approximations of cloud-atmosphere interactions and works for any view point and direction, ensuring continuous transitions in appearance as the viewer moves from ground to space. Among others, our approach supports cloud shadows, light shafts, ocean reflections, and earth shadows on the clouds. The sum of these effects can be visualized at more than 120 frames per second on current graphics processing units.

CCS Concepts: • **Human-centered computing** → **Geographic visualization**; • **Computing methodologies** → *Rendering*; *Mesh geometry models.*

Additional Key Words and Phrases: Real-Time, Rendering, Planet, Terrain, Oceans, Atmosphere, Clouds

Authors' addresses: Florian Michelic; Michael Kenzel, Graz University of Technology, Institute of Computer Graphics and Vision, Graz, 8010, Austria; Karl Haubenwallner, VRVis Research Center, Institute of Computer Graphics and Vision, Vienna, 1220, Austria; Bernhard Kerbl; Markus Steinberger, Graz University of Technology, Institute of Computer Graphics and Vision, Graz, 8010, Austria.

0730-0301/2019/0-ART0
https://doi.org/0000001.0000001_2

1 INTRODUCTION

Today, real-time graphics applications for visual entertainment, such as recent installments of the *Battlefield* or *Grand Theft Auto* series, often feature explorable landscapes that span multiple kilometers in size. Games like *No Man's Sky* or *Star Citizen* take it another step further and let players delve into an entire virtual universe, consisting of a seemingly infinite number of individual planets.

Rendering at the planetary scale brings with it many issues to overcome. Fully detailed terrain data can no longer be held in memory in its entirety and simply rendering the terrain geometry as a whole is not an option. While out-of-core solutions exist [Cignoni et al. 2003], they do not handle fast camera transitions well. Localized approaches commonly apply clipmaps to planetary terrain and sample the height field with decreasing granularity as the distance from the viewer grows [Clasen and Hege 2006; Dimitrijević and Rančić 2015]. However, these approaches require a restrictively high grid resolution to avoid terrain morphing and aliasing artifacts during camera movement. We propose a new method for rendering the terrain of a spherical planet that minimizes morphing artifacts even with a low number of grid points and show how the same method can be used to render large bodies of water, such as oceans.

Another essential factor for the realistic appearance of a planetary environment is its atmosphere. Approaches like that of Bruneton and Neyret [2008] can render realistic skies very fast. However, these methods are usually based on a clear sky model and ignore overcast or cloudy scenarios. We extend the method by Bruneton and Neyret to incorporate clouds for realistic atmospheric lighting.

2 TERRAIN AND OCEAN RENDERING

To render our terrain, we first create a coarse persistent grid on the xy plane centered at the planet's center, with coordinates in the range $[-1, 1]$, warp it to resemble a hemisphere and translate it

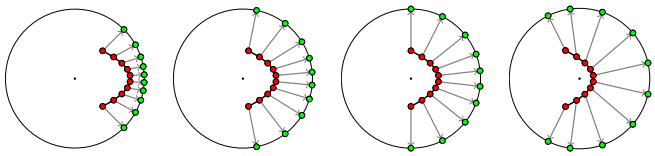


Fig. 2. Projection of the warped grid to the surface as the offset z_s decreases.

along the z-axis by a global offset z_s. We then rotate the grid to face the camera, project it to the ground level of the planet and displace the projected grid vertices to the corresponding elevation in the height field. The offset parameter z_s can be used to control what portion of the planet's surface the grid should cover (see Figure 2). For terrains with a high variance in elevation data, we can compute an adequate z_s that generates vertices even beyond the horizon to ensure the consistent visibility of hills and mountaintops rising up in the distance. To achieve this, we calculate z_s according to

$$z_s = \frac{R^2 + d^2 - (h+s)^2}{2r \cdot (h+s)}$$

where r is the radius of the planet's ground level, R is the distance from the planet center to the highest elevation, d is the distance from the planet center to the camera position, $h = \sqrt{d^2 - r^2}$ and $s = \sqrt{R^2 - r^2}$. During rendering, we perform view frustum culling and tessellation of the grid based on the screen size of the grid cells.

Naïvely generating a new grid centered at the current camera position in each frame would result in unsightly morphing artifacts [Clasen and Hege 2006]. To alleviate this issue, we maintain a proxy position $\mathbf{p}_s$ close to the camera for which we compute the grid. $\mathbf{p}_s$ is snapped to select locations such that new sample points coincide with as many sample points of previous frames as possible.

To render a planet's oceans, we can use the same method as for the terrain and project a grid onto the water surface level. However, instead of sampling a height field, we calculate the displacement $\mathbf{p}'$ of a grid vertex $\mathbf{p}$ at animation time t using the Gerstner wave functions with $\mathbf{p}' = \mathbf{Gerstner}(\mathbf{p}, t, w)$, where $w = \{\mathbf{d}_i, A_i, \omega_i, \varphi_i, Q_i\}$ are artistic parameters, as defined by [Finch 2004]. However, since this approach assumes a planar surface and causes artifacts when applied directly to a sphere, we introduce *spherical Gerstner wave functions* as an extension: We first select a wave origin for each layered wave function on the planet's surface and define the unit vector $\mathbf{o}_i$ as pointing from the center of the planet towards that wave origin. We then calculate the distance l_i from the vertex $\mathbf{p}$ to the wave origin along the surface. The wave direction d_i lies on the tangent plane at $\mathbf{p}$ and points in the direction opposite of the wave origin. With $\mathbf{v} = \frac{\mathbf{p}}{||\mathbf{p}||}$, the equations for d_i and l_i at position $\mathbf{p}$ are:

$$d_i = \mathbf{v} \times ((\mathbf{v} - \mathbf{o}_i) \times \mathbf{v}) \qquad l_i = \arcsin\left(\frac{\|\mathbf{v} - \mathbf{o}_i\|}{2}\right) 2r$$

3 ATMOSPHERE RENDERING

The sky's appearance is typically affected by three major contributions: the sun, the planet's atmosphere, and clouds. To efficiently include these factors as part of the rendering equation, Bruneton and Neyret [2008] introduced a precomputation schema that allows calculation of the transmittance $T_A(\mathbf{p}_1, \mathbf{p}_2)$ and in-scattering

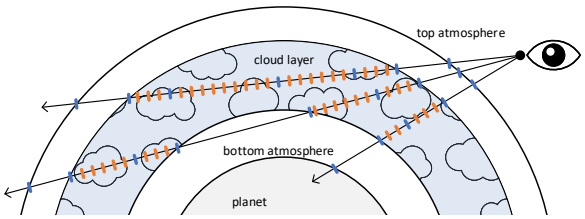


Fig. 3. Overview of our combined atmosphere and clouds ray marching approach. We split the atmosphere into three distinct regions and perform adaptive ray marching, with separate sample positions for the transmittance and in-scattering values for clouds (orange) and the atmosphere (blue).

$S_A(\mathbf{p}_1, \mathbf{p}_2)$ of the atmosphere between any two points, provided the atmosphere is constant across the entire planet. We extend their method by introducing dynamic clouds, modeled as density values in 3D textures and rendered by performing adaptive ray-marching. We first divide the atmosphere into three concentric layers: an upper layer, containing only the atmospheric effects above the clouds, a cloud layer where atmosphere and clouds interact, and a lower layer, containing again only atmospheric effects. We then shoot rays from the camera position in the current viewing direction. Rays traverse through the individual layers and terminate once they reach either the planet surface or outer space. We compute T_A and S_A for the upper and lower layer once between their entry and exit points. Within the cloud layer, we perform ray-marching and assume linear changes in S_A between any two points. For the contribution of clouds, we use a smaller step size to sample their local density and combine their influence on transmittance and in-scattering with the interpolated S_A and applied T_A from the atmosphere (see Figure 3).

4 CONCLUSION

We have presented a method that allows for terrain, ocean and unified cloud-atmosphere rendering in real time for a system that ensures consistency and continuity for transitions from ground to space. The combination of these techniques yields a feasible and holistic approach for the rendering of entire virtual worlds. Of course, apart from terrain, ocean and sky, it takes more details to render a realistic planet, such as fauna, flora, roads and cities to name just a few. However, the procedural generation or integration of these things is a different challenge. Nevertheless, since we are able to render highly challenging images with many cloud samples, light shafts and complex cloud shadows at more than 120 fps, our solution leaves sufficient room for potential future extensions.

REFERENCES

Eric Bruneton and Fabrice Neyret. 2008. Precomputed Atmospheric Scattering. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*. 1079–1086.

P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. 2003. Planet-sized batched dynamic adaptive meshes (P-BDAM). In *IEEE Visualization, 2003. VIS 2003.* 147–154. https://doi.org/10.1109/VISUAL.2003.1250366

Malte Clasen and Hans-Christian Hege. 2006. Terrain Rendering Using Spherical Clipmaps. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*. 91–98.

Aleksandar M. Dimitrijević and Dejan D. Rančić. 2015. Ellipsoidal Clipmaps – A planet-sized terrain rendering algorithm. *Computers & Graphics* (2015), 43–61.

Mark Finch. 2004. Chapter 1. Effective Water Simulation from Physical Models. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, Randima Fernando (Ed.). Pearson Higher Education, 5–29.