# Real-Time Rendering of Procedurally Generated Planets

Florian Michelic[*]
*Supervised by: Michael Kenzel*[†]

Institute of Computer Graphics and Vision
Graz University of Technology
Graz / Austria

## Abstract

We present a simple and efficient solution for seamless and distortion-free representation of planetary terrain, ocean waves, and volumetric clouds solely based on precomputation and on-the-fly calculations, running in real time. Our approach to terrain and ocean mesh generation relies on projecting a persistent grid onto the surface of the planet that takes level-of-detail and frustum-culling into account. We modify a planar wave function to account for the curved surface of the planet, creating seamless, evenly spaced waves across the planet, while remaining fully controllable and adjustable in order to reflect the various dynamics of ocean waves. We further show how we render volumetric clouds along with precomputed atmospheric scattering to properly integrate the clouds into the atmosphere. Our method allows for reasonable cloud-atmosphere interaction and accounts for all viewpoints and viewing directions with continuous transition from ground to space.

**Keywords:** Real Time, Planet, Terrain, Ocean, Atmosphere, Clouds

## 1 Introduction

Rendering large landscapes has always been a difficult task for computer graphics. It requires the representation of a variety of natural phenomena such as terrain, water and the sky. The size of the scene and the freedom of movement of the camera plays a major role in the implementation of those effects. Terrain data can no longer be stored completely in memory and rendering the entire geometry in each frame quickly becomes inefficient as the scene grows. Algorithms such as geomipmapping [4] and geometry clipmaps [12] rely on level-of-detail (LOD) and view-frustum culling to render terrain as detailed as possible in real time. However, most of such algorithms are based on a flat surface as input and can not simply be applied to the curved surface of a planet. The surface of a planet is three-dimensional, which makes placing geometry and sampling textures more difficult. Since the camera should be able to move from ground to space, it is also necessary that effects such as atmospheric scattering and clouds are designed for all viewing directions and camera positions. Our contributions are:

- We propose a new projection method for the persistent grid mapping algorighm [11] that eliminates the problem of morphing vertices.
- We extend the Gerstner wave function [7] to a spherical domain.
- We introduce a new method for unified cloud-atmosphere rendering that allows for continous transition from ground to space.

## 2 Related Work

**Planetary Terrain** The non-planar surface of the planet makes rendering terrain more difficult in addition to LOD and frustum-culling requirements. A common approach is to adapt existing algorithms to the curved surface of the planet.

Clasen et al. [3] apply the GPU-based geometry clipmaps of Asirvatham et.al. [1] to planetary terrain. They use nested concentric rings instead of regular grids to achieve a better mapping to the sphere. Their solution does not account for frustum-culling. Dimitrijević et al. [6] introduce Ellipsoidal Clipmaps (ECM), another way to apply the principle of geometry clipmaps to planetary terrain. They divide the surface of the planet into three seamlessly stiched partitions, one equatorial and two polar. The grid is generated on-the-fly. Their approach enables efficient caching and fast data streaming with reduced data preprocessing.

Mahsman J. [9] extends the persistent grid mapping algorithm (PGM) [11] to create planetary terrain. The algorithm combines ray casting with rasterization, running completely on the GPU. A persistent grid is projected via ray-sphere intersections from screen space to world space onto the surface of a sphere, i.e., the planet's ground level. The adapted PGM algorithm allows for rendering of large terrain datasets providing continuous LOD. A major drawback of using PGM for terrain rendering is visible morphing of vertices as the projected grid moves with the camera, a

[*]florian.michelic@student.tugraz.at
[†]michael.kenzel@icg.tugraz.at

problem which we solve by using a different projection method.

**Planetary Wave Modeling** S. J. Li et al. [10] use PGM to create an input mesh for ocean wave modeling. Here, PGM is a good choice as the moving ocean surface hides the morphing vertices. They use the longitude and latitude of the vertices, denoted as $\alpha$ and $\beta$, to sample a two dimensional wave noise function. They propose to scale $\alpha$ with $\cos(\beta)$ in order to account for the distortion towards the poles. However, after re-implementing this mapping, we found that it suffers from other distortions as well as seam artifacts as illustrated in Figure 1. De-lie et al. [5] generate the ocean mesh with the geomipmapping algorithm [4] and use the method in [15] for wave modeling. They divide the global ocean surface into blocks for which multi-frame height fields are precomputed and stored on disk, trading space for time. A quad tree is used for scene culling and selecting tiles with appropriate resolution to be stored into buffers and rendered, only specific areas are generated in real time. Our approach, on the other hand, does not require any special considerations to go towards enabling LOD and still offers quite some flexibility in terms of wave modeling, all without significant preprocessing and using only very little memory.

**Clouds and Atmospheric Scattering** Schneider et al. [14] model clouds with precomputed, tiling 3D textures and render them through ray marching. Two textures define the shape of the clouds while a third 2D texture is used to add distortion. Lighting accounts for Beer's law, the Henyey-Greenstein phase function [8] and a powder effect that yields dark edges. Several optimizations are introduced to speed up rendering. The major performance gain relies on using a quarter resolution buffer to update the final image over several frames.

Bruneton et al. [2] formulate a rendering equation for atmospheric scattering that allows for precomputation. The transmittance and scattered light is precomputed into textures for all view and sun directions. The precomputed data allows for an efficient computation of the transmittance and in-scattered light between two points within the atmosphere at runtime. Their approach reproduces several effects such as daylight and twilight sky color, aerial perspective and also allows shadows to be taken into consideration to produce light shafts. Our contribution is to combine both approaches to allow for unified atmosphere and clouds rendering with real-time performance.

## 3 Grid Projection

We create terrain in two steps. First, we create a mesh on the planet's ground level, i.e., the surface of a sphere. Then we offset the vertices from the planet's ground level to create planetary terrain. We evaluate 3D Simplex noise [13] to get the terrain height for a certain position on the planet's

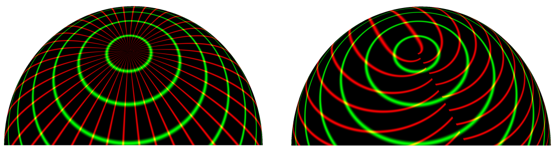
Figure 1: left: Using $(\alpha, \beta)$ results in compressed waves, where $-\pi \leq \alpha \leq \pi$ is the longitude and $-\frac{\pi}{2} \leq \beta \leq \frac{\pi}{2}$ the latitude. right: Using $(\alpha \cdot \cos(\beta), \beta)$ removes compressed waves but also introduces distortion and seams.

ground level. This is benefitial as we do not have to deal with the curved surface of the planet. We can simply layer different frequencies of the noise function to create procedural terrain. The mesh generation on the planet's ground level is more difficult as it must provide LOD and frustum-culling to achieve real-time performance. As with PGM [9], we use a persistent grid and project it to the planet's ground level, but we assume the grid to be in world space instead of screen space. Figure 2 illustrates the initial world-space grid. The center of the planet is located at the origin of the world-space coordinate system. We place the grid at the planet's center and apply a global offset $z_s$ along the z-axis. Then we project the vertices to the planet's ground level with

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \frac{\mathbf{g}}{\|\mathbf{g}\|} \cdot r, \tag{1}$$

where $\mathbf{v}$ is the resulting vertex position, $r$ is the radius of the planet, and $\mathbf{g} = \begin{bmatrix} g_x & g_y & g_z \end{bmatrix}^T$ the original grid vertex coordinate. The offset $z_s$ allows us to control the projection but causes some issues as illustrated in Figure 3. For $z_s > 0$, the projected mesh provides more detail at the horizon which is not desired. Moving the grid closer to the planet center increases the error. For $z_s = 0$, the grid does no longer cover any area and for $z_s < 0$ the back faces of the mesh are projected to the planet's ground level. As a solution, we choose to displace the grid along the z-axis to approximate a hemisphere before applying $z_s$. This certainly solves all projection issues as illustrated in Figure 4.

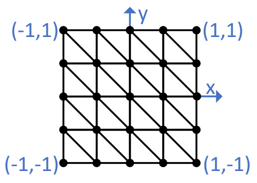

Figure 2: Initial world-space grid layout illustrated with $5 \times 5$ vertices.

We use

$$g_z = (1 - g_x^n)(1 - g_y^n), \tag{2}$$

with $n = 4$ to displace the initial grid to an approximated hemisphere. However, $n = 2$ also works but we found that $n = 4$ results in a better projection for $z_s < 0$. The global offset $z_s$ defines the area that the grid covers on the

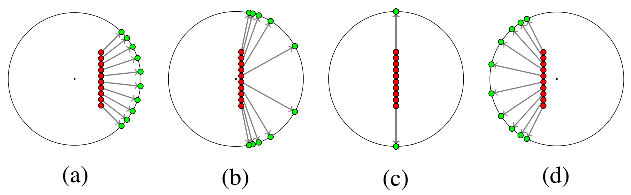

Figure 3: Grid projection approach. (a, b) initial grid (red) projected to the planet's ground level (green) with positive offset $z_s > 0$. (c) no offset (d) negative offset.

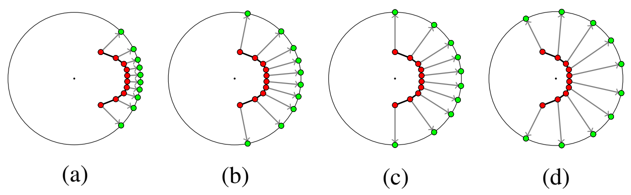

Figure 4: Grid projection approach with approximated hemisphere due to Equation 2. The projected grid covers less space near the horizon and for $z_s < 0$ the grid is correctly projected beyond the hemisphere of the planet.

planet's ground level. For example, if the camera is on the ground, $z_s$ should be positive because there is only a fraction of the planet visible to the camera. As the camera moves into space, $z_s$ should decrease because more of the planet becomes visible. The offset calculation also relates to further displacement of the projected mesh. We need to take this displacement into account when we calculate $z_s$ as high mountains may rise beyond the horizon. We assume the worst case where the highest possible mountain is currently beyond the horizon and just the peak of it is visible. Figure 5 illustrates the calculation of $z_s$. The distances $h$ and $s$ are given as

$$h = \sqrt{d^2 - r^2} \qquad s = \sqrt{R^2 - r^2},$$

where $d$ is the distance from the planet center to the camera position and $R$ is the distance from the planet center to the highest possible mountain. From $\sin\alpha = \frac{r}{d}$ and $\sin\varphi = \frac{(h+s)\cdot r}{Rd}$ we get the distance $r_s$ with

$$r_s = \frac{y}{\sin\varphi} = \frac{1}{\frac{(h+s)\cdot r}{Rd}} = \frac{Rd}{r\cdot(h+s)}.$$

From $\cos\varphi = \frac{R^2+d^2-(h+s)^2}{2Rd}$ and $r_s$ we compute the offset $z_s$

$$z_s = \cos\varphi \cdot r_s = \frac{R^2+d^2-(h+s)^2}{2r\cdot(h+s)}. \qquad (3)$$

The calculation of $z_s$ only accounts for the camera distance $d$ and assumes the camera to be on the positive z-axis. Therefore, we must rotate the grid to face the camera. For a given camera position $\mathbf{p}$ and an arbitrary linearly independent vector $\mathbf{b}$, we use a rotation matrix $\mathbf{R}$ with

$$\mathbf{w} = \frac{\mathbf{p}}{\|\mathbf{p}\|}, \qquad \mathbf{v} = \frac{\mathbf{w}\times\mathbf{b}}{\|\mathbf{w}\times\mathbf{b}\|}, \qquad \mathbf{u} = \mathbf{w}\times\mathbf{v}$$

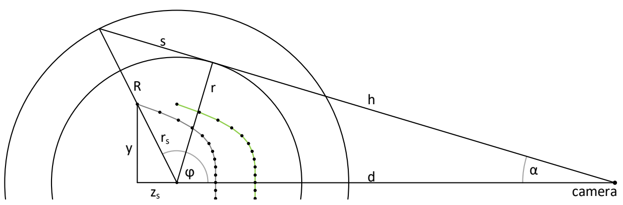

Figure 5: A grid (green) is displaced by $z_s$ to capture every possible visible terrain beyond the horizon. The radius $r$ is the planet radius at ground level and $R$ is the distance from the planet center to the highest mountain.

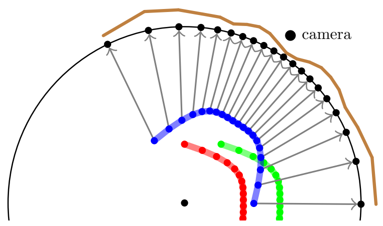

Figure 6: Grid projection. (red) initial grid displaced to approximated hemisphere. (green) grid shiftet by $z_s$. (blue) grid rotated to face the camera. (brown) vertices displaced from the planet's ground level to model terrain.

$$\mathbf{R} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}. \qquad (4)$$

The vector $\mathbf{b}$ needs to be constant and not related to the camera orientation. This allows us to prevent visible morphing of vertices for camera rotations compared to PGM. However, $\mathbf{b}$ must be changed if it is not linearly independent to $\mathbf{p}$, although we never had problems using $\mathbf{b} = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$. Figure 6 illustrates the complete projection approach.

As the grid scales with $d$ and always faces $\mathbf{p}$, morphing still takes place when the camera translates. To eliminate the effect for translations, we keep the camera position for the projection constant until the distance to the actual camera position exceeds a certain threshold. This allows the camera to move freely within a restricted space without any visible morphing. Therefore, we reduce the effect of morphing vertices completly to popping, which happens everytime we update the camera position used for the projection algorithm. We can also prevent this by using opportunities to update it earlier when the change in geometry is less visible, e.g., when looking at the sky.

There is no fustum-culling so far as the projection only adapts to the camera position. However, we apply frustum-culling during tessellation on the GPU by discarding all triangles for which all vertices fall outside a slightly widened camera frustum.

## 4 Ocean Wave Modeling

Using the projection method described above, we generate a mesh with an appropriate tessellation for ocean wave simulation. Our approach is based upon the Gerstner wave function [7] that creates directional and circular waves on a planar surface. The vertices are displaced along the z-axis by the sum of sine waves and also along the x,y-plane to create sharp waves with the nice property that vertices are concentrated at the crests. Furthermore, it allows easy control over wave properties such as amplitude and direction and suits well for real-time applications. We define a mapping to apply the principle of Gerstner waves to the curved surface of the planet which we further refer to as spherical Gerstner waves. The Gerstner wave function is given as

$$P_g = \begin{bmatrix} \mathbf{p}_x + \sum_{i=1}^{N} \mathbf{d}_{i,x} Q_i A_i \cos\left(\omega_i \mathbf{d}_i \mathbf{p} + \varphi_i t\right) \\ \mathbf{p}_y + \sum_{i=1}^{N} \mathbf{d}_{i,y} Q_i A_i \cos\left(\omega_i \mathbf{d}_i \mathbf{p} + \varphi_i t\right) \\ \sum_{i=1}^{N} A_i \sin\left(\omega_i \mathbf{d}_i \mathbf{p} + \varphi_i t\right), \end{bmatrix} \tag{5}$$

where $N$ is the number of waves, $\mathbf{p}$ is the input vertex, $A_i$ is the amplitude of wave $i$, $\mathbf{d}_i$ is the direction vector along which the wave travels, $\omega_i$ is the frequency, $\varphi_i$ controls the wave speed and $Q_i$ is a steepness parameter that controls the sharpness of the crests. For circular waves, $\mathbf{d}_i$ is calculated with

$$\mathbf{d}_i = \frac{\mathbf{p} - \mathbf{c}_i}{\|\mathbf{p} - \mathbf{c}_i\|}, \tag{6}$$

where $\mathbf{c}_i$ defines the wave centers. In contrast to directional waves, circular waves map very good to planets as for a wave with $\mathbf{c}_i = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, the dot product $\mathbf{d}_i \cdot \mathbf{p}$ in Equation 5 is $\|\mathbf{p}\|$ and $\mathbf{d}_i = \frac{\mathbf{p}}{\|\mathbf{p}\|}$. We define spherical Gerstner waves with a unit vector $\mathbf{o}_i$ that points to the origin of the waves on the planet. Figure 7 illustrates the parameters. The vector $\mathbf{o}_i$ points to the origin of the waves. The distance $l_i$ is the distance from the origin to the vertex. The direction vector $\mathbf{d}_i$ at the vertex lies on the tangent plane and points towards $-\mathbf{o}_i$. The vector $\mathbf{v}$ is the normalized vertex position. Waves start at the origin and meet up at the opposite side of the planet, i.e., where $-\mathbf{o}_i$ points to. We calculate $\mathbf{d}_i$ and $l_i$ at vertex position $\mathbf{p}$ with

$$\mathbf{d}_i = \mathbf{v} \times ((\mathbf{v} - \mathbf{o}_i) \times \mathbf{v}) \qquad l_i = \arccos(\mathbf{v} \cdot \mathbf{o}_i) \cdot r, \tag{7}$$

where $\mathbf{v} = \frac{\mathbf{p}}{\|\mathbf{p}\|}$ and $r$ is the distance from the planet center to the desired sea level. The distance $l_i$ is technically only another representation of the latitude and since we do not include the longitude, our method is free of distortions and seams compared to [10]. We split up the horizontal and vertical displacement in Equation 5 and replace $\mathbf{d}_i \cdot \mathbf{p}$ with $l_i$ yielding the spherical Gerstner wave function $P_s$ as

$$\begin{aligned} P_s = \mathbf{v}r + \mathbf{v} \sum_{i=1}^{N} \left(A_i \sin(\omega_i l_i + \varphi_i t)\right) \\ + \sum_{i=1}^{N} \left(Q_i A_i \cos(\omega_i l_i + \varphi_i t) \cdot \mathbf{d}_i\right). \end{aligned} \tag{8}$$

We calculate the tangent space vectors in a similar way. With Gerstner waves, the normal vector is calculated with

$$\mathbf{n}_g = \begin{bmatrix} -\sum_{i=1}^{N} \mathbf{d}_{i,x} A_i \omega_i \cos\left(\omega_i \mathbf{d}_i \mathbf{p} + \varphi_i t\right) \\ -\sum_{i=1}^{N} \mathbf{d}_{i,y} A_i \omega_i \cos\left(\omega_i \mathbf{d}_i \mathbf{p} + \varphi_i t\right) \\ 1 - \sum_{i=1}^{N} Q_i A_i \omega_i \sin\left(\omega_i \mathbf{d}_i \mathbf{p} + \varphi_i t\right) \end{bmatrix}. \tag{9}$$

As with Equation 8, we replace $\mathbf{d}_i \cdot \mathbf{p}$ with $l_i$ and use $\mathbf{d}_i$ and $\mathbf{v}$ to map $\mathbf{n}_g$ to the tangent plane of $\mathbf{p}$ such that

$$\begin{aligned} \mathbf{n}_s = \mathbf{v} - \mathbf{v} \sum_{i=1}^{N} Q_i A_i \omega_i \sin\left(\omega_i l_i + \varphi_i t\right) \\ - \sum_{i=1}^{N} \mathbf{d}_i A_i \omega_i \cos\left(\omega_i l_i + \varphi_i t\right). \end{aligned} \tag{10}$$

The tangent vector is constant for a single direction from the wave origin as $\mathbf{n}_s$ always lies on the plane defined by $\mathbf{d}_i$ and $\mathbf{v}$. Therefore, we calculate the tangent vector $\mathbf{t}_s$ and the associated bitangent $\mathbf{b}_s$ with

$$\mathbf{t}_s = \sum_{i=1}^{N} \left( \frac{\mathbf{d}_i \times \mathbf{v}}{\|\mathbf{d}_i \times \mathbf{v}\|} \right) \qquad \mathbf{b}_s = \mathbf{n}_s \times \mathbf{t}_s.$$

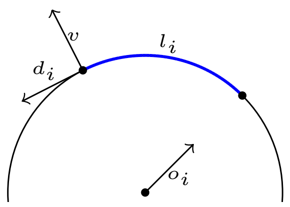

Figure 7: Spherical Gerstner wave parameters.

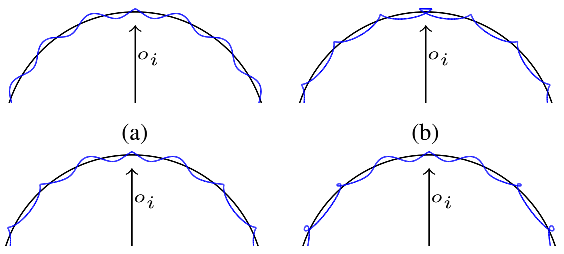

Figure 8: Spherical Gerstner waves. (a) $Q_i = 0$ yields the usual sine wave. (b) $Q_i = 1$ gives sharp crests but causes loops at $\mathbf{o}_i$ and $-\mathbf{o}_i$. (c) Scaling $Q_i$ to 0 removes the loops. (d) Further increasing of $Q_i$ causes loops at the crests.

Results for Equation 8 are illustrated in Figure 8. The displacement along $\mathbf{d}_i$ on the tangent plane of $\mathbf{p}$ causes a small offset from the sea level which allows $Q_i$ to exceed 1 without causing loops as seen in Figure 8b. While this is true for Equation 8, it does not apply to the normal calculation in Equation 10. Therefore, $Q_i$ should never exceed 1 as in [7]. Another problem arises at $\mathbf{o}_i$ and $-\mathbf{o}_i$. The vertices are displaced beyond the orgin and form loops as illustrated in Figure 8b. A simple solution is to fade out the displacement along $\mathbf{d}_i$ towards $\mathbf{o}_i$ and $-\mathbf{o}_i$ by scaling $Q_i$ to 0. This is applied using the smoothstep function as

$$Q_i' = Q_i \cdot \text{smoothstep}(1 - |\mathbf{v} \cdot \mathbf{o}_i|, e_0, e_1), \tag{11}$$

where $e_0$ and $e_1$ should be choosen such that $e_1 > e_0 > 0$ to fade out $Q_i$ early enough with proper transition across several wave fronts.

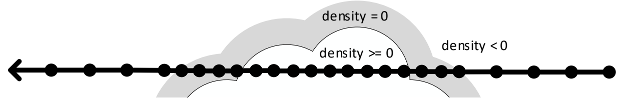

Figure 9: Ray marching through the cloud texture peforming either a small or large step. The choosen step size only depends on the sampled density represented as black dots.

## 5 Clouds

**Modeling** We model clouds by precomputing Simplex [13] and Worley [16] noise into a single $256^3$, 16-bit float texture representing cloud density. Values $\geq 0$ model cloud material whereas values $< 0$ indicate that there are no clouds. As in [14], we perform either small or large ray-marching steps but we use a different selection criteria. Rather than taking a step back or switching back to large steps, we precompute a border of zero density around each cloud that matches our large step size. This way, the sampled density immediately tells us to perform either a small (density $\geq 0$) or a large (density $< 0$) step as illustrated in Figure 9. This saves us a few instructions and allows to stop or continue ray marching efficiently which we do frequently with our unified cloud-atmosphere rendering approach.

We define a cloud layer with a top and bottom radius and tile the cloud texture in world space across the planet as illustrated in Figure 10. Ray marching is only performed within the cloud layer. We use a height signal similar as in [14] to fade out the density to zero near the top and bottom layer. The texture coordinate $\mathbf{c}$ for a given world-space position $\mathbf{p}$ is given as

$$\mathbf{c} = \operatorname{fract}\left(\mathbf{p}\frac{1}{w}\right), \tag{12}$$

where $w$ is the tile width in world space. The height signal $h$ is computed with

$$h = 1 - \left(2 \cdot \frac{\operatorname{clamp}(\|\mathbf{p}\| - l_b, 0, l_t - l_b)}{l_t - l_b} - 1\right)^2, \tag{13}$$

where $l_t$ is the distance from the planet center to the top layer and $l_b$ the distance to the bottom layer. The density for a given world-space position $\mathbf{p}$ is then given as

$$\operatorname{density}(\mathbf{p}) = \begin{cases} -1 & \text{if} \quad \operatorname{tex}(\mathbf{c}) < 0 \\ h \cdot \operatorname{tex}(\mathbf{c}) & \text{else,} \end{cases} \tag{14}$$

where $\operatorname{tex}(\mathbf{c})$ is the linear interpolated voxel at texture coordinates $\mathbf{c}$ of our precomputed cloud texture.

**Cloud Lighting** Let $T_C$ be the transmittance of the clouds between two points $\mathbf{p}_0$ and $\mathbf{p}_1$ within the atmosphere. Light that arrives at $\mathbf{p}_1$ and travels towards $\mathbf{p}_0$ is partially out-scattered due to clouds. The transmittance defines how much light arriving at $\mathbf{p}_1$ reaches $\mathbf{p}_0$. We compute $T_C$ by ray marching from $\mathbf{p}_0$ to $\mathbf{p}_1$. For each density sample, we accumulate $T_C$ with

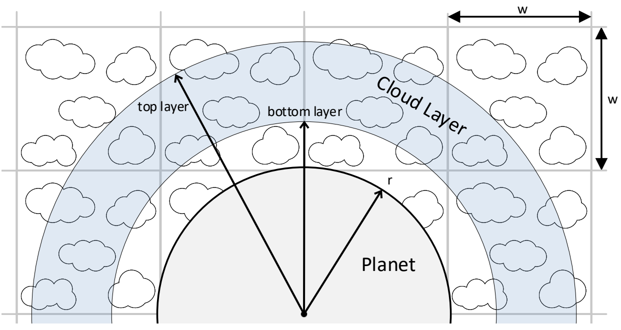

Figure 10: Planet with cloud layer. The precomputed texture is tiled across the planet in world space. The cloud layer is defined with a top and bottom radius, i.e., the top and bottom cloud layer.

$$T_{C,0} = 1, \quad T_{C,n} = T_{C,n-1} \cdot e^{-E \cdot \Delta \cdot d}, \tag{15}$$

where $E$ is the extinction factor, $\Delta$ is the step size in world space and $d$ is the density sample. Along to the transmittance, we compute the in-scattered light $I_C$ of the clouds between the two sample points. That is, the amount of light that is scattered towards $\mathbf{p}_0$ between $\mathbf{p}_0$ and $\mathbf{p}_1$. We accumulate the in-scattered light with

$$I_{C,0} = 0, \quad I_{C,n} = I_{C,n-1} + S \cdot HG \cdot \Delta \cdot d \cdot I_{sun} \cdot T_{C,n}, \tag{16}$$

where $S$ is the scattering factor, $HG$ the Henyey-Greenstein phase function [8] and $I_{sun}$ the incident light. We perform six density samples along a ray towards the sun and compute the transmittance $T_{sun}$ according to Equation 15. The incident light $I_{sun}$ is then given as $I_{sun} = T_{sun} \cdot L_{sun}$, where $L_{sun}$ is a constant sun radiance.

**Atmospheric Scattering** We apply precomputed atmospheric scattering as proposed by Bruneton et. al. [2]. This allows us to get the transmittance $T_A$ and the in-scattered light $I_A$ of the atmosphere. We can either fetch $T_A$ and $I_A$ between a point and the top atmosphere boundary when looking at the sky, or between two points within the atmosphere. When looking at the sky, we just display $I_A$. When looking at the ground, we display $I_A + T_A \cdot R_G$, where $R_G$ is the ground radiance, i.e., the result of terrain or ocean shading.

For view rays that intersect with clouds, there is also an in-scattered light $I_C$ along the view ray towards the camera. However, if we evaluate Equations 15 and 16 from the camera along the view ray to its end point (e.g. terrain) and sample the atmosphere for the same distance, we have no basis for combining $I_C$ and $I_A$ as $T_C$ and $T_A$ provide no information about the position of the clouds. Of course, since clouds and atmosphere influence each other, it makes a difference where the clouds are located along the view ray. The precomputation of atmospheric scattering whereas assumes a clear sky and does not accout for any clouds.

We integrate the clouds into the calculation of the atmosphere at a later stage by making several atmosphere samples along the view ray. We can divide the atmosphere into an arbitrary sized number of slices but need to accumulate the total $T_A$ and $I_A$ along the whole view ray ourselves. In other words, we perform ray marching on the atmosphere itself, by simply sampling the in-scattered light and the transmittance of the precomputed textures for each step. This can be written as

$$T_{A,0} = 1, \quad T_{A,n} = T_{A,n} \cdot T_{A,n-1} \tag{17}$$

$$I_{A,0} = 0, \quad I_{A,n} = I_{A,n-1} + I_{A,n} \cdot T_{A,n-1}. \tag{18}$$

Without clouds, we would get the same result as if we were sampling the atmosphere once. However, with multiple samples, we can evaluate Equations 15 and 16 separately for each atmosphere step and let the atmosphere and cloud transmittances influence each other's in-scattered light. Figure 11 illustrates multiple atmosphere samples along the view ray. For each atmosphere sample, we compute $T_C$ and $I_C$ and accumulate them together with $T_A$ and $I_A$ into a combined transmittance $T$ and a combined in-scattered light $I$. The combined transmittance $T$ is computed with

$$T_0 = 1, \quad T_n = T_{A,n} \cdot T_{C,n}, \tag{19}$$

where the index n of $T_{C,n}$ indexes the result of Equation 15 between two subsequent atmosphere sample points. For $I$ we assume that $T_C$ affects $I_A$ but $I_C$ is only indirectly affected by $T_A$ via $T_{n-1}$ which can be written as

$$I_0 = 0, \quad I_n = (I_{A,n} \cdot T_{C,n} + I_{C,n}) \cdot T_{n-1}. \tag{20}$$

**Rendering** To render the atmosphere, we evaluate Equation 20 along the view ray. We omit $T_C$ and $I_C$ in Equation 20 for atmosphere samples outside the cloud layer. Equations 15 and 16 yield $T_C = 1$ and $I_C = 0$ due to $d = 0$. We start ray marching at the camera position and end either at the sky or at the ground. View rays may intersect the cloud layer once, twice or only the top or bottom cloud layer, i.e., when the camera is within the cloud layer. We only consider rays that intersect the cloud layer and apply the atmosphere otherwise as without clouds. Figure 12 illustrates ray marching through the cloud layer when the camera is in space. We sample the atmosphere once from the top atmosphere to the top cloud layer. We continue with small, increasing steps towards the bottom cloud layer or the top cloud layer. From there, we either sample the sky radiance, the ground radiance or hit the bottom cloud layer again.

For atmosphere samples within the cloud layer we need to compute $T_C$ and $I_C$ between two sample positions. We start a ray marcher that evaluates Equations 15 and 16 between the two atmosphere sample points. The resulting transmittance $T_C$ and in-scattered light $I_C$ of the clouds for this segment is then accumulated according to Equation 20.

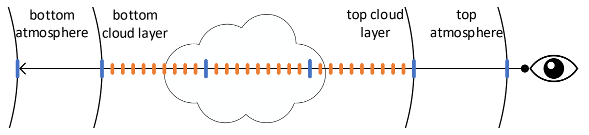

Figure 11: Unified atmosphere-clouds ray marching, illustration of 5 subsequent atmosphere slices along the view ray (blue). For each slice within the cloud layer, we ray-march the clouds separately in smaller steps (orange).

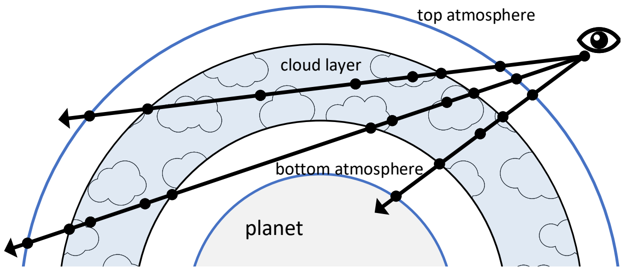

Figure 12: Atmosphere and clouds ray marching. All rays through the cloud layer with camera position in space, i.e., outside the top atmosphere boundary. We omit rays that only touch the top cloud layer.

## 6 Results

We implemented our solution with OpenGL on a NVIDIA GeForce GTX1060, all images are rendered in full-HD. Figure 13 illustrates our planet rendering approach. We precompute Simplex noise into a $1024^2$ texture to define the shape of the continents using a longitute-latitude mapping. We use this texture along with on-the-fly evaluation of Simplex noise to displace the projected grid from the planet's ground level. We compute per-vertex normals by sampling the height of the terrain at two neighboring points. We then shade the terrain with the Phong illumination model and use a three-planar projected normal map to get detail normals. Performance results for terrain rendering can be found in Table 1 with reference images in Figure 14. We use $100 \times 100$ vertices and 5 noise layers for all other renders as this seems enough to model terrain and popping is barely visible due to tessellation. The change in terrain geometry (popping) is illustrated in Figure 15.

For ocean rendering we project a $80 \times 80$ grid, apply tessellation and then displace the vertices using our spherical Gerstner wave function. Shading accounts for the underwater terrain and either atmosphere reflections or screen-space reflections of the terrain. The performance for increasing numbers of waves can be found in Table 2. Figure 16 illustrates the result for 20 waves and also shows the circular and directional appearance of a single wave.

As a final step, we render the clouds along with the atmosphere as a post-processing effect. The two textures containing the world-space positions of the terrain and the ocean determine whether the ray ends up at the ground or at the sky and the color textures provide us with the ground

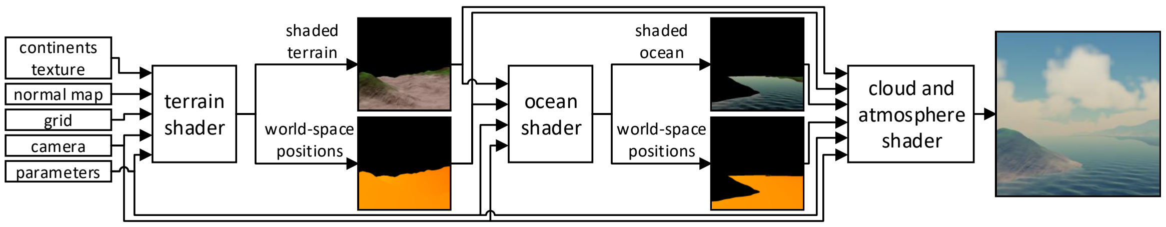

Figure 13: Planet rendering.

radiance $R_g$. For each $R_g$ we compute a transmittance towards the sun with one density sample in the middle of the cloud layer and use it to darken $R_g$ for cloud shadows. Figure 17 illustrates cloud-atmosphere interaction for varying atmosphere parameters and Figure 18 shows the planet viewed from space.

| noise layers | $50 \times 50$ | $100 \times 100$ | $200 \times 200$ |
|---|---|---|---|
| continents | 0.8 ms | 0.5 ms | 0.9 ms |
| 1 | 1.0 ms | 0.7 ms | 1.3 ms |
| 5 | 1.3 ms | 1.3 ms | 1.9 ms |
| 10 | 1.7 ms | 2.6 ms | 2.6 ms |

Table 1: Terrain performance for increasing noise layers and grid resolutions. See Figure 14 for reference images.

| 20 | 50 | 100 | 1000 |
|---|---|---|---|
| 0.6 ms | 0.7 ms | 0.9 ms | 4.0 ms |

Table 2: Spherical gestner waves performance for 20, 50, 100 and 1000 waves for view in Figure 16.

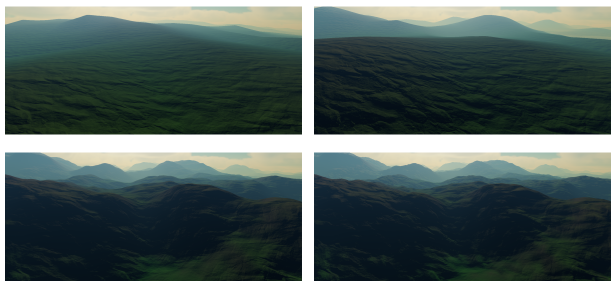

Figure 14: Reference images for Table 1 with initial grid resolution $100 \times 100$ vertices. From top-left to bottom-right, continents texture only, 1, 5 and 10 noise layers.

## 7 Discussion

In contrast to PGM for planetary terrain [9], our approach to mesh generation does not suffer from visible morphing of vertices. While this is our main improvement, it should also be mentioned that our projection method requires only a few lines of shader code. The offset calculation already accounts for terrain beyond the horzion which is important for small planets with relatively high terrain. PGM whereas requires a special sampling camera to capture possible visible terrain beyond the horizon. Our mapping of Gerstner waves to the curved surface of the planet does not suffer from distortion or seams as the mapping proposed in [10] and requires only little memory compared to [5]. Each wave consumes 28 Bytes of GPU memory which is in total 560 Bytes for 20 waves. This is significantly less than the precomputed multi-frame height fields for the global scope in [5] and there is also no need to store ocean data on disk and upload them to the GPU at runtime.

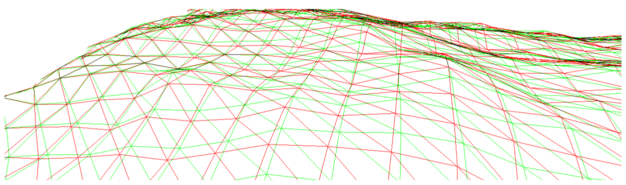

Figure 15: Overlay of previous (red) and current (green) terrain mesh for raw projected grid.

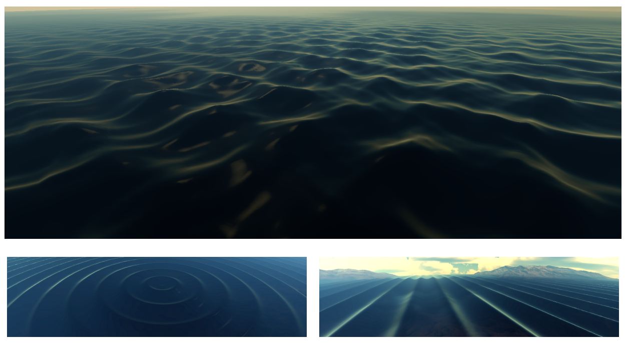

Figure 16: The view (top) used for ocean performance measurements in Table 2. Circular waves at the origin (bottom-left) and directional waves far away from the origin (bottom-right).

In the future we may investigate different tessellation strategies along with potential on-the-fly switching of the persistent grid. We may further improve procedural modeling and shading to account for various wheater and atmospheric conditions for a more reasonable representation of real world planets. There are also potential ways to speed up cloud ray marching. This could involve LOD textur-

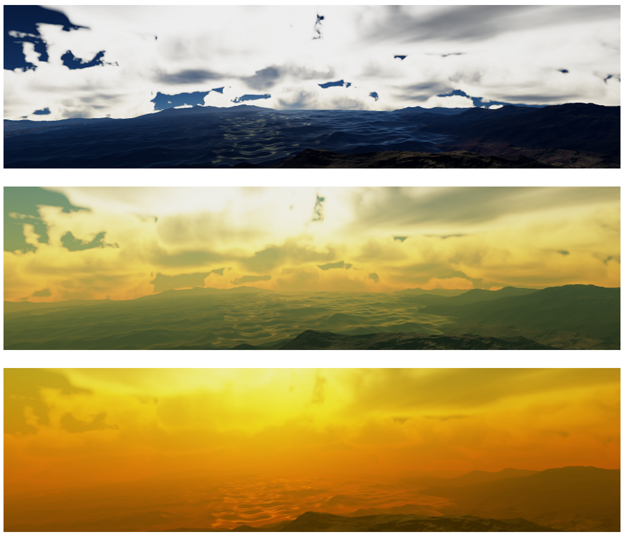
Figure 17: Interaction of cloud and atmosphere for varying atmosphere parameters. Rendered with 70 fps including 2.96 ms for terrain, 0.26 ms for ocean and 10.98 ms for atmosphere and clouds.

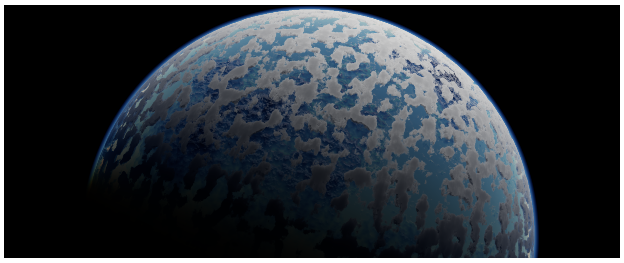
Figure 18: Planet viewed from space.

ing using a downsampled version of our cloud texture or sharing incident light samples across ray-marching steps.

## 8 Conclusion

We presented a new method for terrain, ocean and unified cloud-atmosphere rendering that allows for real-time planet rendering providing continuous transition from ground to space.

## References

[1] Arul Asirvatham and Hugues Hoppe. Chapter 2 terrain rendering using gpu-based geometry clipmaps. In Matt Pharr and Randima Fernando, editors, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*, pages 27–46. Addison-Wesley Professional, 2005.

[2] Eric Bruneton and Fabrice Neyret. Precomputed atmospheric scattering. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*, pages 1079–1086, 2008.

[3] Malte Clasen and Hans-Christian Hege. Terrain rendering using spherical clipmaps. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*, pages 91–98, 2006.

[4] Willem H. de Boer. Fast terrain rendering using geometrical mipmapping, 2000.

[5] M. De-lie, G. Peng-fei, and W. Mi. On realization of visualization system for global ocean simulation. In *2010 International Conference on Audio, Language and Image Processing*, pages 1446–1450, 2010.

[6] Aleksandar M. Dimitrijević and Dejan D. Rančić. Ellipsoidal clipmaps – a planet-sized terrain rendering algorithm. *Computers & Graphics*, pages 43–61, 2015.

[7] Mark Finch. Chapter 1. effective water simulation from physical models. In Randima Fernando, editor, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, page 5–29. Pearson Higher Education, 2004.

[8] L. C. Henyey and J. L. Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, 93:70–83, 1941.

[9] Mahsman J. Projective grid mapping for planetary terrain. Master's thesis, 2010.

[10] S. J. Li and X. S. Zhan. Modeling and rendering of ocean battlefield scenes. In *2011 Workshop on Digital Media and Digital Content Management*, pages 40–44, 2011.

[11] Yotam Livny, Neta Sokolovsky, Tal Grinshpoun, and Jihad El-Sana. A gpu persistent grid mapping for terrain rendering. *The Visual Computer*, pages 139–153, 2008.

[12] Frank Losasso and Hugues Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. In *ACM SIGGRAPH 2004 Papers*, pages 769–776, 2004.

[13] Ken Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 681–682, 2002.

[14] Andrew Schneider. The real-time volumetric cloudscapes of horizon: Zero dawn. SIGGRAPH2015 Advances in Real-Time Rendering in Games course, 2015.

[15] Jerry Tessendorf. Simulating ocean water. 1999.

[16] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 291–294, 1996.