Real-Time Rendering of Planets with Atmospheres

Tobias Schafhitzel Martin Falk Thomas Ertl

Visualization and Interactive Systems, Universität Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{schafhitzel|falkmn|ertl}@vis.uni-stuttgart.de

ABSTRACT

This paper presents a real time technique for planetary rendering and atmospheric scattering effects. Our implementation is based on Nishita's atmospheric model which describes actual physical phenomena, taking into account air molecules and aerosols, and on a continuous level-of-detail planetary renderer. We obtain interactive frame rates by combining the CPU bound spherical terrain rendering with the GPU computation of the atmospheric scattering. In contrast to volume rendering approaches, the parametrization of the light attenuation integral we use makes it possible to pre-compute it completely. The GPU is used for determining the texture coordinates of the pre computed 3D texture, taking into account the actual spatial parameters. Our approach benefits from its independence of the rendered terrain geometry. Therefore, we demonstrate the utility of our approach showing planetary renderings of Earth and Mars.

Keywords: Atmospheric scattering; Ray Tracing; Planets; Terrain Rendering; Multiresolution; GPU Programming;

1 INTRODUCTION

Realistic image synthesis plays a crucial role in modern computer graphics. One topic is the light scattering and absorbtion of small particles in the air, called atmospheric scattering. In the last years, several methods were developed to simulate this effect. These methods make it possible to simulate light beams[4], the sky[8;15](including the colors of sunrise and sunset) and the Earth viewed from space[12]. Also, the application area of these methods is very broad, reaching from educational programs, CAD applications and terrain representations to driving, space or flight simulations in games. However, all these methods have to overcome the high computational costs, caused by the complexity of solving the scattering integral.

In this paper, we discuss a fast and precise method for physically based image synthesis of atmospheric scattering. Exact integration is provided by a pre computation step, where the whole scattering integral is solved. This unloads the graphics hardware in a way, that the GPU is only used for transforming the actual spatial parameters to fetch the pre computed values. In contrast to volume rendering approaches, we simulate the atmospheric scattering effect from each point of view by rendering only two spheres.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright UNION Agency – Science Press, Plzen, Czech Republic.

First, the light intensity reaching the observer's eye is pre computed for each point inside and outside the atmosphere. This is followed by a fast evaluation of the scattering integral for each rendered vertex easily by fetching the pre computed texture. This fast computation makes this method applicable to rendering scenes consisting of a larger amount of vertices, as they occur in terrain rendering. Therefor, we prove our method by combining atmospheric scattering and a continuous level of detail planetary terrain renderer. Due to the fact, that the pre computed light scattering values are valid for planar spheres only, applying a rough planet's surface would implicate several problems. Thereby, the integration distances change depending on the planet's structure, as well as the light contribution of the terrain itself, which has to be accounted additionally. To address this issues, we discuss the correct use of the pre computed lookup texture using a minimal number of shader instructions.

2 PREVIOUS WORK

Since physical phenomena are one of the most interesting research topics in computer graphics, there has been a considerably large amount of work on atmospheric scattering in the last years. Most of the earlier work are based on ray tracing, which is an appropriate method for the creation of photo-realistic representations of the atmosphere. But due to the high computational costs, an interactive visualization was not realizable this time. In previous work[7;10;18], the scattering and absorbtion of light is discussed. In 1993, Nishita et al.[12] discussed the basic equations of Rayleigh and Mie scattering effects. In this work, the attenuation of the incident light was considered at each point of the atmosphere as well as the attenuation from this point to the viewer. Further-

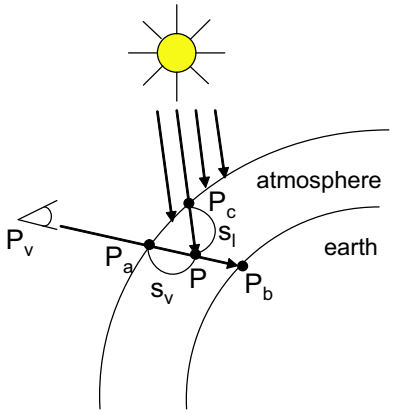

Figure 1: Calculation of light reaching P_v.

more, additional to the sky's color, the colors of clouds and sea were discussed. Another model for rendering the sky in the daytime was proposed by Preetham[15] and Hoffman[6], a physics based night model was described by Jensen[8]. An extension of Preethams's model was proposed by Nielsen[11] by considering a varying density inside the atmosphere. In 2002, Dobashi et al.[4] proposed a GPU based method to implement atmospheric scattering using a spherical volume rendering. They solved the discrete form of the light scattering integral by sampling it with slices. Depending on the shape of the slices, which are planar or spherical, atmospheric scattering effects as well as shafts of light, as occurring between clouds can be visualized. Later, in 2004, O'Neil[13] proposed an interactive CPU implementation, solving the scattering integral by ray casting. This method avoids expensive volume rendering by representing the atmosphere by only two spheres, one for the outer and one the inner boundary of the atmosphere. Ray casting is used to solve the discrete form of the scattering integral by separating the ray into segments and calculate the attenuation of the light and to the viewer for each sample point. As the connection between the vertices of the two spheres and the viewer is used as view ray, this approach strongly depends on the complexity of the scene. A vertex shader implementation was published by O'Neil[14].

One of the main topics in terrain visualization consists of the application of continuous level of detail methods[5;9;16]. Cignoni et al.[1] as well as Röttger et al.[17] used a quadtree based approach to achieve an adaptive mesh refinement of the terrain data. An adaption of the BDAM approach for the application of planetary height fields was published by Cignioni[2]. In our approach, the method of Roettger is extended for planetary terrain rendering.

3 ATMOSPHERIC SCATTERING

In this section, the basic equations of Rayleigh and Mie scattering are considered (according to Nishita[12]). First, we discuss the scattering of air molecules, described by the Rayleigh scattering equation:

$$I_p(\lambda,\theta) = I_0(\lambda) K \rho F_r(\theta)/\lambda^4, \tag{1}$$

which describes the light scattered in a sample point P, in dependence of λ, the wavelength of the incident light at P and θ, the scattering angle between the viewer and the incident light at P. I_0 stands for the incident light, ρ for the density ratio (given by $\rho = \exp(\frac{-h}{H_0})$), depending on the altitude h and scale height $H_0 = 7994m$, and F_r for the scattering phase function, which indicates the directional characteristic of scattering ($F_r = \frac{3}{4}(1+\cos^2(\theta))$).

K describes the constant molecular density at sea level

$$K = \frac{2\pi^2(n^2-1)^2}{3N_s}, \tag{2}$$

with N_s, the molecular number density of the standard atmosphere and n, the index of refraction of the air.

In equation 1, the relation between the scattered light and the wavelength of the incident light describes the strong attenuation of short wavelengths. This is due to the wavelength's inversely proportional behavior to the light attenuation. To determine the light intensity reaching P_v in Figure 1, we have to consider two steps. First, for each point P between P_a and P_b, the light reaching this point needs to be attenuated. Second, the resulting light intensity on each point P needs to be attenuated a second time on its way to the viewer at P_v. The attenuation between two points inside the atmosphere is determined by the optical length, which is computed by integrating the attenuation coefficient standing for the extinction ratio per unit length, along the distance s_v. The attenuation coefficient β is given by:

$$\beta = \frac{8\pi^3(n^2-1)^2}{3N_s\lambda^4} = \frac{4\pi K}{\lambda^4}, \tag{3}$$

which is integrated along the distance S and yields

$$t(S,\lambda) = \int_0^S \beta(s)\rho(s)ds = \frac{4\pi K}{\lambda^4}\int_0^S \rho(s)ds. \tag{4}$$

The scattering of aerosols is described with a different phase function. Therefor, the improved Henyey-Greenstein function by Cornette[3] is used. To adjust the rate of decrease of the aerosol's density ratio, the scale height H_0 has to be 1.2km[19]. The optical length of aerosols is the same as for air molecules, except for the $\frac{1}{\lambda^4}$ dependance. According to Nishita[12], if the equations above are applied to Equation 1, the light intensity at P_v leads to:

$$I_v(\lambda) = I_s(\lambda)\frac{KF_r(\theta)}{\lambda^4}\cdot \int_{P_a}^{P_b} \rho \exp(-t(PP_c,\lambda) - t(PP_a,\lambda))ds. \tag{5}$$

4 REAL-TIME ATMOSPHERIC SCATTERING

In current approaches for computing atmospheric scattering effects, the performance mainly depends on the complexity of the scene. Therefor, the scattering integral of Equation 5 is solved for each vertex belonging to the scene geometry. By applying modern graphics hardware, this approaches are quite fast, especially when planets are restricted to simple spheres, consisting only of few vertices. But an increasing complexity of the scene makes it impossible to compute the scattering integral for each vertex in real time anymore. This is the case if you intend to render the structure of planets. We avoid this lack of performance, by sourcing out the scattering integral. Therefor, we pre compute the scattering integral and store it in a 3D texture, and thereby, we reduce the number of instructions which are necessary to obtain the light attenuation value. The following sections describe the creation of the lookup texture and its use at the rendering stage.

4.1 Creating the Scattering Texture

As we aim to pre compute the light scattering integral, we have to reconsider Equation 5. It defines the light contribution reaching the observers eye, when it is placed at a position P_v and his view ray penetrates the atmosphere from the point P_a to the point P_b. Basically, the light is attenuated two times, for each point on $\overline{P_aP_b}$. The first time from the light source to a position P on the view ray, and the second time from P to the observer's position P_v. In accordance with this observation, the resulting light contribution depends on four variables: the observer's position P_v, the position of the light source P_c and the entry and exit position of the atmosphere P_a and P_b. Furthermore, approximating the integral as a Riemann sum requires an additional sample variable P. If we naively pre compute the scattering integral, we have to compute the light intensity for each point in the atmosphere, looking in each direction. This means that we have to consider several distances $\overline{P_aP_b}$. And we also have to keep in mind, that all this pre computations have to be done for each position of the light source. This would result in nine scalar values when the distance $\overline{P_aP_b}$ is represented by a three dimensional vector.

O'Neil[13] suggested to simplify the computation of the optical depth (Eq. 4) from the light source to P by parameterizing each point P by its height and its angle to the Sun. This simplifies Equation 5 enormously, because the pre computed values can be used to determine the optical depth from the light source to the sample point $t(PP_c,\lambda)$ as well as for the attenuation from the sample point to the observer. Nevertheless, the integral from P_v to P remains. For dealing with this expensive computation, we have extended O'Neils approach by additionally considering the view direction.

The algorithm works as follows: first, we define the parametrization of our 3D texture. As discussed above, we have to consider a number of parameters, which have to be reformulated in a way they can be used to parameterize a three dimensional lookup table. For the sake of simplicity, we now assume the observer to be situated inside the atmosphere, and discuss the general case later in this section. Basically, the observer can be situated at an arbitrary position, looking in an arbitrary direction at an arbitrary daytime. Considering the observer's position P_v and his view direction R_v at a specific daytime we can assume that if $P'_v = (0,|P_v|,0)^T$ and $R'_v = (sin(\theta),cos(\theta),0)^T$ then

$$cos(\theta) = \frac{1}{|P_v||R_v|} < P_v, R_v >= \frac{1}{|P'_v|} < P'_v, R'_v > . \quad (6)$$

This means, that each actual position and the corresponding view direction can be described by its height $h = |P_v|$ and the view angle θ. Exploiting this behavior, we place the camera at each height inside the atmosphere to send out the view rays in each direction. Based on the fact, that the boundaries of the atmosphere are considered as spherical, also the distance $\overline{P_aP_b}$ is the same for every view ray with angle θ with respect to the current position P'_v. While computing $\overline{P_aP_b}$, it is quite important to test R'_v for intersection with the inner and the outer boundary sphere. We also have to consider, that the pre computation regards the planet's surface as a simple sphere. How to deal with structured surfaces is described in Section 4.2.

To complete this formulation, we have to introduce a light source to our model. As discussed above, the attenuation from a light source to an arbitrary position can also be described by the height of the sample point and the angle δ to the light source. For solving equation 5 for a position P'_v and a view ray R'_v, the light attenuation $t(PP_c,\lambda)$ and $t(PP_a,\lambda)$ can easily be computed by sampling along R'_v and computing the values according to the height of the sample point and its angle to the light source. Finally, we introduce the daytime to our model, by applying the pre computation for all angles to the sun. Thus, the angle δ builds the third parameter of the 3D texture.

Special consideration should be taken for the case, when the observer is situated outside the atmosphere. Since there is no light scattering outside the planet's atmosphere, the distance of the observer to the planet is not accounted in our pre computation step. Nevertheless, viewing the planet from outside builds a special case, in which the computation can be considered the same for each position, even if the camera is situated on the atmosphere's outer boundary or the camera is far away. Thus, the camera has to be virtually moved towards the planet, until it hits the outer boundary of the atmosphere. Afterwards, we start the computation. Thus, we only need to consider one additional

height in our pre computation: the height if the camera is outside the atmosphere, which the height of the atmosphere plus an additional, minimal offset.

```
   // Loop over all view angles to the
      camera
1  foreach angleViewer < resZ do
   |  // Get angle θ
2  |  θ = GetViewAngle(angleViewer);
   |  // Generate a view vector
3  |  R'_v = vec3d(sin(θ), cos(θ), 0);
   |  // Loop over all view angles to the
   |     light source
4  |  foreach angleSun < resY do
   |  |  // Get angle δ
5  |  |  δ = GetLightAngle(angleSun);
   |  |  // Loop over all heights of the
   |  |     camera
6  |  |  foreach height < resX do
   |  |  |  // Get current height inside
   |  |  |     the atmosphere
7  |  |  |  h = fRadIn + ((fRadOut − fRadIn) ·
   |  |  |  height)/(resX − 1);
   |  |  |  // Generate the position
   |  |  |     vector
8  |  |  |  P_v = vec3d(0, h, 0);
   |  |  |  // Finally, compute the light
   |  |  |     scattering
9  |  |  |  color = ComputeScattering(P_v, δ, R_v);
10 |  |  end
11 |  end
12 end
```

The code sample above shows how simple the 3D lookup texture is created. The texture is given with its sizes in the x,y and z direction. For each voxel, the corresponding angles and the height is used for computing the light intensity, by evaluating the light scattering integral. Therefor, we solve the discrete form of Equation 5:

$$I_v(\lambda) \quad = \quad I_s(\lambda)\frac{KF_r(\theta)}{\lambda^4}\sum_{i=0}^{k}\rho\exp(-t_l - t_v) \qquad (7)$$

Due to the fact that the whole scattering integral is pre comuted, the sample rate k as well as the sample rate for the optical depth t_l and t_v can be set very high.

4.2 Applying the Scattering Texture

Since the lookup texture is computed, the calculation of the light intensity is quite simple and needs only few instructions on the GPU. During the rendering phase, two independent scene objects are drawn: a sphere, representing the sky and the planet's surface. In Section 5, the rendering of the terrain is discussed in more detail. To demonstrate how the light intensity influences our scene, we first consider the rendering of the sky.

Actually, the sky is represented by rendering only one tesselated sphere, which is placed nearby the outer boundary of the atmosphere. Furthermore, we enable front face culling to ensure, that the ray between the observer and the sphere is penetrating the atmosphere before the intersection. In contrast to the atmosphere, the terrain is rendered with back face culling enabled, because we are only interested in the visible part of the planet's surface. To obtain the light contribution, first the view ray $R_v = P_g - P_v$ is computed, where P_g stands for the position of the current vertex. If the observer is outside the atmosphere, the camera is moved to the outer boundary. Then, the height is obtained by $h = |P_v|$ as well as the cosine of the view angle $cos(\theta) = \frac{1}{|P_v||R_v|} < R_v, P_v >$ and the sun angle $cos(\delta) =< P_c, P_v >$. After rescaling this three parameters to $[0,1]$, the lookup texture is fetched. Because of the nonlinear behavior of the scattering function, it makes sense to implement the texture fetch as a fragment program, instead of a vertex program, with trilinear interpolation enabled. This minimizes the interpolation error, since the sample frequency of the fragment shader is much higher and much faster than a comparable vertex shader implementation.

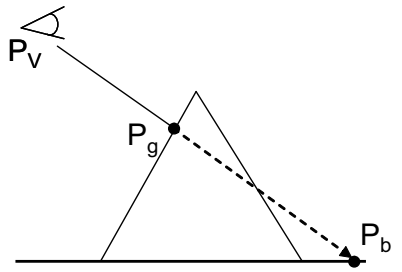

Figure 2: Obtaining the wrong light intensity: instead of $\overline{P_gP_v}$ the distance $\overline{P_vP_b}$ was used for the pre computation

In contrast to the sky, the computation of the terrain needs some more consideration. If we simply apply the texture lookup to any other vertices of the scene geometry, like the vertices representing the terrain, the computation fails. Figure 2 shows this case. This behavior results from the pre computation, which treats the ray to intersect a simple sphere without considering the height field of the terrain. Thus, the light scattering is computed for the whole distance $\overline{P_vP_b}$. To discuss how it is possible to compute the light scattering along the distance $\overline{P_gP_v}$, the pre computed values have to be analyzed. The pre computed light scattering along $\overline{P_vP_b}$ can be formulated as the scattering along $\overline{P_gP_v}$ plus the scattering along $\overline{P_gP_b}$. Rewriting Equation 5 to obtain the light scattering along $\overline{P_gP_v}$ would lead to

$$I'_v(\lambda) = I_s(\lambda)\frac{KF_r(\theta)}{\lambda^4}\cdot \qquad (8)$$
$$\left(\int_{P_v}^{P_b}\rho\exp(-t_l - t_v) - \int_{P_g}^{P_b}\rho\exp(-t_l - t_v)\right),$$

where both terms can be obtained fetching the pre computed texture. In fact, the first term is obtained anyway, simply using the current parameters as input. The second light contribution can be determined easily by moving the camera to the intersection point P_g without changing the view angle. Then we access the lookup texture a second time and subtract the result from the color value of the first texture lookup. By using this mechanism, it is possible to obtain the light contribution for any point inside the atmosphere considering several kinds of geometry.

Finally, we discuss the correct illumination of the terrain. As we are able to compute the light scattering between the object and the viewer, the contribution of the illuminated terrain has been not considered yet. Similar to the air molecules and aerosols, the light illuminating the terrain is attenuated two times. While the attenuation from the light source to the terrain is even the same, the spectral shift of the illuminated terrain to the observer needs to be discussed in more detail: first, we only consider the light intensity I_g reaching the terrain geometry. Mathematically it is defined as

$$I_g(\lambda) = I_s(\lambda)\frac{KF_r(\theta)}{\lambda^4}\cdot\rho\exp(-t(P_gP_c,\lambda))ds. \quad (9)$$

The resulting I_g is used as incident light intensity for illuminating the terrain geometry. We have applied a Lambert reflection, which considers the cosine of the angle φ between the incident light and the terrain normal as the intensity of the light absorbed by the terrain. Introducing the diffuse reflection to equation 9 yields

$$I_g(\lambda) = I_s(\lambda)\frac{KF_r(\theta)}{\lambda^4}\cdot\rho cos(\varphi)\exp(-t(P_gP_c,\lambda))ds. \quad (10)$$

Additionally, the attenuation of the reflected color needs to be attenuated on its way to the viewer. As defined by Nishita [12], the attenuation has to be multiplied with the intensity of the terrain geometry

$$I_{gv}(\lambda) = I_g\exp(-t(P_gP_v,\lambda))ds. \quad (11)$$

This equation describes the light contribution of the terrain, without considering the light scattering along the distance $\overline{P_gP_v}$. Since the light contributions of all sample points are accumulated to determine the intensity reaching the observer, Equation 8 has to be accounted as

$$I''_v(\lambda) = I_{gv} + I'_v, \quad (12)$$

where I''_v stands for the overall intensity reaching the observer's eye if he is looking on the rough surface of a planet. While Equation 8 is implemented as two fetches of the pre computed 3D texture, the easiest way to obtain I_g and I_{gv}, is to adopt the 2D lookup texture described in O'Neil [13]. As this texture stores the optical depth for each point inside the atmosphere, the light attenuation between the light and the geometry can fetched as well as the attenuation between the geometry and the observer. For the other parameters shader constants are used, excepting the phase function $F_r(\theta)$ which is pre computed as 1D texture. The following pseudo fragment shader code demonstrates the low number of instructions used for this complex computation.

```
   // Compute R_v
1  R_v = normalize(P_g − P_v);
   // If the camera is outside the
      atmosphere, move it to the outer
      boundary
2  dist = Intersect(P_v, R_v, SphereOut));
3  if dist > 0 then
4  |  P_v = P_v + R_v · dist;
5  end
   // Compute h, θ and δ
6  h = MapToUnit(|P_v|);
7  θ = MapToUnit(< P_v, R_v >);
8  δ = MapToUnit(< P_c, P_v >);
   // Get the light intensity without
      considering the height field
9  I_v = tex3D(texPre3D, vec3d(h, δ, θ));
   // Move the camera to the intersection
      point to obtain the offset
10 h' = MapToUnit(|P_g|);
11 δ' = MapToUnit(< P_c, P_g >);
12 I_off = tex3D(texPre3D, vec3d(h', δ', θ));
   // Correct the intensity
13 I'_v = I_v − I_off;
   // Now compute the light contribution
      of the terrain
14 F_r = tex1D(texPhase, θ);
15 t_gc = tex2D(texPre2D, h', δ');
16 t_gv = tex2D(texPre2D, h', MapToUnit(< P_g, R_v >));
17 I_gv = I_s K F_r 1/λ^4 · ρ · < N_g, P_c > ·exp(−t_gc − t_gv);
   // Finally, get the overall light
      intensity at P_v
18 I''_v = I_gv + I'_v;
19 return I''_v;
```

Please keep in mind, that in the case the camera is situated inside the atmosphere, the computation of t_{gv} requires one additional lookup (see O'Neil [13]).

5 PLANETARY TERRAIN RENDERING

In this section we discuss the terrain renderer we have optimized for the visualization of round shapes, like planets. Therefor, we extended an existing planar terrain renderer [17] to render spherical objects. Indeed, the extension to achieve the round shape of a planet is quite easy, simply applying a spherical mapping of the vertices. However, this modification implicates a number of further necessary adaptations.

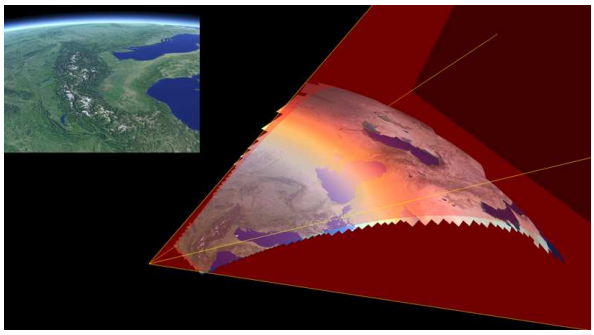

Figure 3: A two stage clipping algorithm is applied on the whole planet. Upper left: the corresponding image reaching the observer.

One adaption consists of the dynamic mesh refinement (geomorphing). As defined in [17], the mesh refinement criterion f is defined by:

$$f = \frac{l}{d \cdot C \cdot \max(c \cdot d2, 1)}, \tag{13}$$

where l is the distance to the viewer and d is the length of the block, which needs to be refined. The constant values C and c are standing for minimum global resolution and the desired global resolution. The surface roughness value is defined as $d2 = \frac{1}{d} \max |dh_i|$, where dh_i is the elevation difference between two refinement levels. If $f < l$, the mesh needs to be refined. Applying this criterion to a spherical terrain representation, the viewpoint and the vertices needs to be in the same coordinate system. To obtain a correct length l, the spherical transformation of the terrain needs to be attended (see Figure 4(i)). Therefore, if we assume the position P_v of the viewer given in world space, an inverse spherical mapping of P_v is necessary.

For further optimization, we introduced a spherical view frustum clipping: the terrain consists of several tiles, which can differ in the resolution of their local height field, but not in their size in world space. Thus, a tile with a higher local resolution implicates a higher level of refinement in world space. Due to their constant size, these tiles are well suited as input for the clipping algorithm. The clipping algorithm consists of two stages: in the first stage, all non-visible tiles are clipped. This stage is accomplished before the mesh refinement, by a comparison of the vector of the view direction and the four vertices, building the border of the tile to be tested. If the cosine between this vectors is negative, the tile can be considered as visible. Thus, the first stage can be considered like a kind of crude back face culling, just working with whole terrain tiles. The second stage implements a smoother clipping, taking into account the grid refinement. Therefor, the quad tree (for further information see [17]), in which the geometry is stored, is traversed down and each vertex, situated in the center of the current quad, is tested against the four clipping planes. If the visibility test fails, the quad is clipped, by removing the entry of the quad tree. Figure 3 shows the two-stage clipping (center) and the resulting visualization (upper left).

domain size	Earth Dataset $257^2 \times 24 \times 12$	Mars Dataset $65^2 \times 24 \times 12$
outside	67.26	83.51
inside	30.31	56.02
only terrain		
outside	76.44	102.37
inside	35.83	61.19

Table 1: Performance outside and inside the atmosphere (in fps).

Resolution	Optimized	Vertex Shader
128^2	608.49	151.06
256^2	181.54	35.92
512^2	48.67	9.97

Table 2: Performance of the atmosphere only, (in fps).

6 RESULTS

Table 1 shows the measured performance in frames per seconds. All the measurements are made on an AMD Athlon64 X2 Dualcore 4800+ 2.4 GHz machine with a GeForce 7900 GT graphics card with 256 MB of memory and a viewport of 800×600. First, the planets are viewed from "outside" the atmosphere, as in Figure 4(a) and (g). The "inside" measurement considers the frames per second when the camera is situated inside the atmosphere, like in Figure 4 (c) and (h). Both data sets are divided into 24 tiles with respect to the longitude and 12 tiles with respect to the latitude. The quadratic value stands for the size of the tile. The resolution of the pre computed 3D texture was chosen with 128^3. This size can be considered as sufficient. Furthermore, it is noticeable, that larger sizes of this texture are not really influencing the performance. If we consider the measurements of Table 1, we can see that the rendering outside the atmosphere is much faster than inside. This is due to the adaptive refinement of the terrain mesh, which is inactive, when the observer's distance to the planet is too large. In this case, only a tesselated sphere is rendered. In contrast to this case, the number of triangles increases extremely if the camera is situated nearby the planet's surface. In order to demonstrate that the resulting frames per second strongly depends on the performance of the terrain renderer, we replaced the 10 tiles representing the focused mountains, with high resolution height fields of 1800×1800 cells. This allows us to increase the number of drawn triangles to compare it with the achieved rendering speed, if

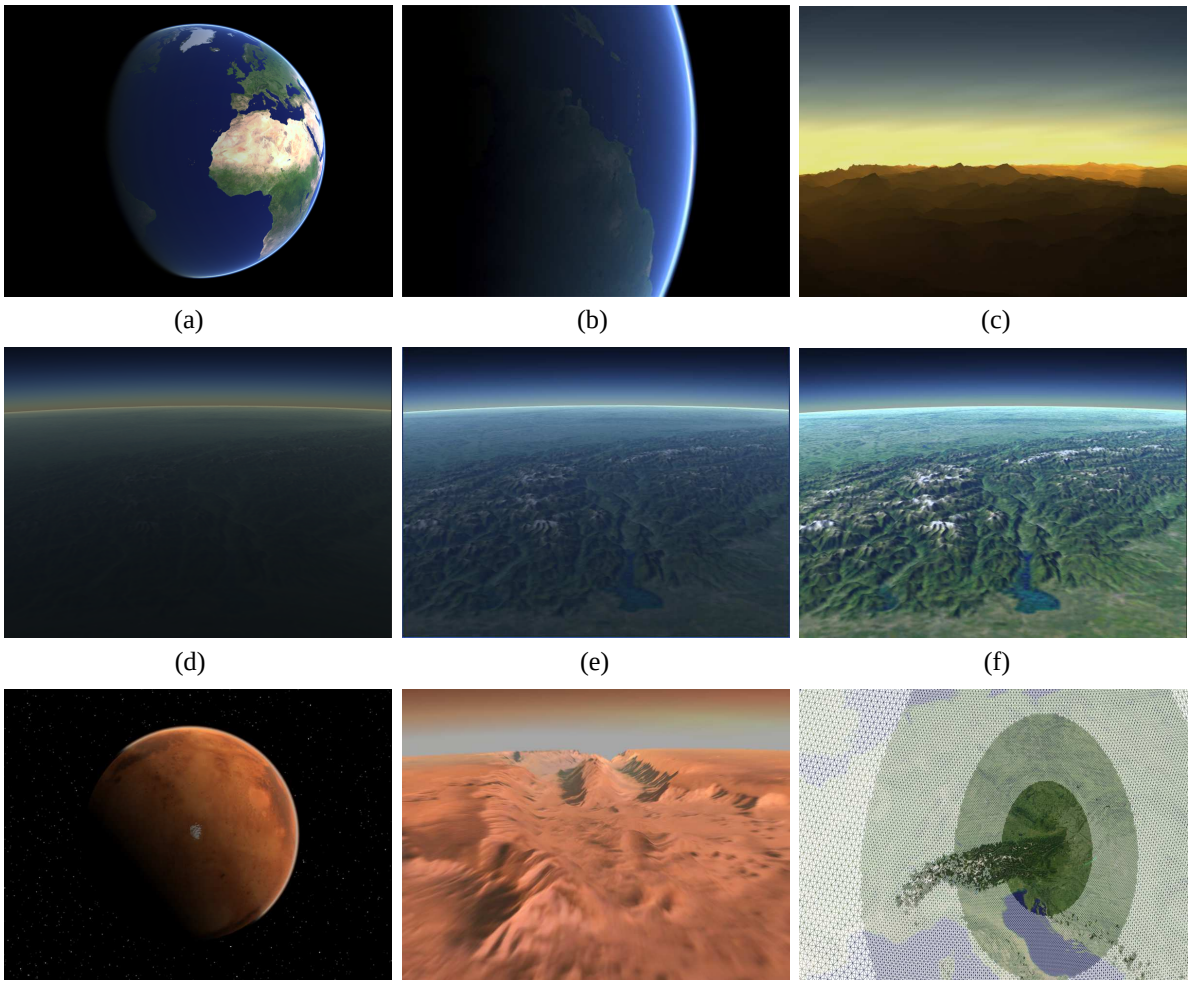

Figure 4: (a) the Earth viewed from space, (b) South America after the sunrise, (c) sunrise over the Alps, (d) sunrise (e) early morning, (f) midday, (g) Mars viewed from space, (h) Valles Marineris with martian atmosphere, (i) adaptive mesh refinement

only the terrain is rendered without any atmospheric effects. The mean value of the rendering power, required for the atmospheric scattering effects, is about 15%.

Table 2 shows the increase in performance, when the evaluation of the scattering integral is optimized by our method. The frames per second of our implementation are compared with a vertex shader implementation. For this measurements, only the atmosphere is rendered applying 2 spheres with the given number of vertices per sphere. The resolution of the pre computed 3D texture is also 128^3. The table shows, that our method is 4 - 5.3 times faster than the vertex shader implementation. It is also perceptible, that this ratio increases proportional to the number of rendered triangles, what can be ascribed to the high number of vertex shader instructions and the costs of vertex texture fetches.

Figure 4 (g) and (h) demonstrate the flexibility of the applied light scattering method, by replacing the Earth's atmosphere with the Martian atmosphere by simply modifying the molecular density and the wavelength of the incident light. Additional dust particles, which mainly influences the color of the Martian atmosphere, are unaccounted. Figure 4 (d), (e) and (f) show the Alps viewed from Italy at different day times, starting with the sunrise, over to the early morning hours to midday. This sequence illustrates the dependency of the light scattering and the angle to the sun. Finally, (i) demonstrates how our spherical refinement mechanism works.

7 CONCLUSION AND FUTURE WORK

We have presented an interactive technique for planetary rendering taking into account atmospheric scattering effects. High efficiency is achieved by combining the CPU based terrain renderer with the atmospheric rendering. Therefor, the complete scattering integral, described by Nishita[12], is evaluated in a separate pre

computation step. This is done by computing the atmospheric scattering for each position inside the atmosphere, parameterized by its height and the angles to the viewer and to the Sun. The results are stored into one 3D texture. The GPU is used to compute the light intensity reaching the observer's eye, regarding the structure of the planet's surface and the correct illumination of the terrain geometry. We have discussed, how the pre computed 3D texture can be used to solve the problems mentioned above. Finally, a planetary terrain renderer was introduced for the adaptive mesh generation and rendering of height fields on spherical objects. The results shows clearly, that the evaluation of the scattering integral is absolutely independent of the scene complexity, what makes it attractive to utilize it with large scale renderings.

In the future, the 3D texture can also be used for rendering other scenes, like snow or rain simulations. It is also thinkable to use it as part of a complete weather simulation or to illuminate large outdoor scenes in games.

8 ACKNOWLEDGMENTS

We would like to thank the NASA for providing the earth textures and the MOLA datasets of Earth and Mars.

REFERENCES

[1] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(2):505–514, 2003.

[2] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Planet-sized batched dynamic adaptive meshes (P-BDAM). In *Proc. IEEE Visualization*, pages 147–155, 2003.

[3] W.M. Cornette and J.G. Shanks. Physical reasonable analytic expression for the single-scattering phase function. *Applied Optics*, 31(16):3152–3160, 1992.

[4] Y. Dobashi, T. Yamamoto, and T. Nishita. Interactive rendering of atmospheric scattering effects using graphics hardware. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 99–107, 2002.

[5] M.H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 135, 1995.

[6] N. Hoffman and A.J. Preetham. Real-time light-atmosphere interactions for outdoor scenes. *Graphics programming methods*, pages 337–352, 2003.

[7] H.W. Jensen and P.H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320, 1998.

[8] H.W. Jensen, F. Durand, J. Dorsey, M.M. Stark, P. Shirley, and S. Premoze. A physically-based night sky model. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 399–408, 2001.

[9] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G.A. Turner. Real-time, continuous level of detail rendering of height fields. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118, 1996.

[10] N.L. Max. Atmospheric illumination and shadows. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 117–124, 1986.

[11] R.S. Nielsen. Real time rendering of atmospheric scattering effects for flight simulators. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2003.

[12] T. Nishita, T. Sirai, K. Tadamura, and E. Nakamae. Display of the earth taking into account atmospheric scattering. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 175–182, 1993.

[13] S. O'Neal. Real-time atmospheric scattering. www.gamedev.net/reference/articles/article2093.asp, 2004.

[14] S. O'Neal. Accurate atmospheric scattering. *GPU Gems*, 2:253–268, 2005.

[15] A.J. Preetham, P. Shirley, and B. Smits. A practical analytic model for daylight. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 91–100, 1999.

[16] E. Puppo. Variable resolution terrain surfaces. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 202–210, 1996.

[17] S. Röttger, W. Heidrich, P. Slusallek, and H.-P. Seidel. Real-time generation of continuous levels of detail for height fields. In *Proc. WSCG '98*, pages 315–322, 1998.

[18] H.E. Rushmeier and K.E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 293–302, 1987.

[19] S. Sekine. Optical characteristics of turbid atmosphere. *J Illum Eng Int Jpn*, 71(6):333, 1992.