High Performance Graphics (2014)
Jonathan Ragan-Kelley and Ingo Wald (Editors)

High-Performance Rendering of Realistic Cumulus Clouds Using Pre-computed Lighting

Egor Yusov[1]

[1] Intel Corporation

Figure 1: Clouds rendered by the proposed algorithm.

Abstract

We present a new method for rendering realistic cumulus clouds in real time. The clouds in our approach consist of randomly rotated and scaled copies of a single reference particle. During the pre-processing, we pre-compute optical depth, single and multiple scattering inside the reference particle for every camera position, orientation and light direction, and store the information in the look-up tables. At run time, information from the look-up tables is used to compute the cloud shading, avoiding any ray marching or slicing. To control the level of detail, we introduce a new technique which provides high fidelity for close clouds while using a coarse representation for distant regions. In addition to this, we present a new method for blending particles. Compared to traditional alpha-blending, this method produces more accurate visual results by accounting for volumetric intersection. The method merges collection of individual particles into a continuous medium, and also eliminates temporal artifacts. Our technique is able to produce realistic images at high frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Rendering realistic clouds has always been a desired feature for a wide range of applications, from military flight simulators to computer games. Clouds consist of a great number of tiny water droplets which interact with the photons traveling through the cloud. The dominant type of interaction is scattering which changes the photon direction. Typically, a photon undergoes many scattering events before it leaves the cloud, and almost all photons entering it eventually exit. Accurately modeling multiple scattering in the cloud is prohibitively expensive, even for off-line renderers. Real-time techniques rely on simplifications to reduce computational complexity, like ignoring multiple scattering or using procedural texturing and other non physics-based methods.

There are numerous kinds of clouds such as cumulus, stratus, cirrus as well as combinations of these. In our work, we concentrate on rendering cumulus (puffy) clouds, which have the most pronounced volumetric nature. Since precisely

modeling light propagation inside the cloud is too expensive to perform in real time, we offset the cost by pre-computing light transport at pre-process. Our main contributions to the state of the art are:

1. Pre-computed solution to optical depth integral inside the inhomogeneous volumetric particle.
2. Pre-computed solution to single and multiple scattering inside the homogeneous spherical particle.
3. Real-time lighting model based on the pre-computed solutions to optical depth and scattering.
4. Method to control the level of detail which enables rendering large outdoor environments with clouds.
5. Technique for volume-aware blending of particles.

The rest of the paper is organized in the following way. In section 2, we give an overview of the previous work and in section 3 present the mathematical background. Section 4 describes our pre-computed solution to optical depth, single and multiple scattering. Implementation details are given in section 5, results and discussion follow in section 6. Section 7 concludes the paper.

2. Related Work

Many cloud rendering algorithms have been developed in the last few years. A recent survey by Hufnagel and Held [HH12] provides a thorough overview. In this section we briefly review the most relevant methods.

The most accurate results come from off-line renderers, which can afford long rendering times (minutes to hours to generate single frame) and can simulate very complex effects. Jensen and Christensen used photon maps to generate high quality images of participating media [JC98]. Nishita et al. modeled an energy exchange between voxels [NDN96] of a uniform grid, and reduced the number of possible paths by taking into account the strong forward scattering.

Some methods completely ignore the physics of the problem so as to render quickly, and rely on different procedural techniques. Gardner used textured ellipsoids to model cumulus clouds [Gar85]. Elinas and Stuerzlinger implemented Gardner's method in real time using the texturing capabilities of the GPU [ES00]. Ebert combined implicit functions with turbulence-based procedural techniques to model and animate clouds [Ebe97]. Schpok et al. presented a system which uses volumetric implicit functions to model cloud shape, and adds fine-grain details using 3D noise [SSEH03]. Rendering is performed using slice-based volume rendering technique, while color is artistically controlled by interpolation between "shadowed" and "lit" colors. Wang described an artist-driven technique which renders clouds as alpha-blended textured sprites [Wan04]. In this method, cloud shape, texturing, and shading are manually modeled by artists. The main advantage of non physics-based approaches is speed. However, creating realistic-looking images requires tweaking different parameters and a lot of artistic effort. Dynamic lighting and cloud animation increase the cost even more, and many effects, such as cloud self-shadowing or silver linings, are hard to model.

Many methods approximate the physical process of cloud formation and lighting using simplifying assumptions. Dobashi et al. [DKY*00] modeled cloud dynamics using cellular automaton. Rendering was performed using splatting, and only single scattering was modeled. Harris and Lastra [HL01] extended this method by simulating multiple forward scattering at pre-process and anisotropic scattering at run time. Clouds and lighting conditions are limited to be fixed in this method. Miyazaki et al. [MDN04] used a half-angle slicing technique introduced by Kniss et al. [KHSM03] to compute single scattering in the cloud. Riley et al. [REK*04] adapted the same technique to approximate multiple forward scattering by using a series of pre-convolved phase functions.

The idea of offsetting run-time rendering costs by performing expensive pre-computations is exploited by many algorithms. Sloan et al. [SKS02] performed an off-line light transport simulation to encode the transfer from incident lighting into outgoing radiance. Transfer functions are represented using low-order spherical harmonics, for every point, which limits the method to low-frequency lighting environments and static objects. It cannot be used for rendering significantly anisotropic scattering media like cloud droplets under direct sunlight.

To efficiently render stratiform clouds, Bouthors et al. pre-computed light transport in a plane-parallel slab, and then fitted the local cloud shape to the slab [BNL06]. This algorithm was then extended to render cumulus clouds [BNM*08]. The method uses the results of an extensive Monte-Carlo simulation fitted into analytic functions to find area on the cloud surface (called collector area) through which the light reaches the shading point. This algorithm is rather involved, computationally expensive, and difficult to reproduce.

Ament et al. described the direct volume rendering method which also exploits pre-integration of light transport to approximate multiple scattering [ASW13]. They examine a finite spherical region of a homogeneous medium and use path tracing to pre-compute the radial distribution of scattered light leaving the sphere. The results of simulation are stored in the look-up table parameterized by the scattering/extinction properties of the medium, anisotropy of the phase function, and incident light direction. The final in-scattering intensity is computed by ray marching. Each sample point is considered the center of an imaginary sphere. Local properties of the volume data inside the sphere are averaged and used as the coordinate to look-up the approximated intensity of the scattered radiance leaving the sphere in the direction of camera. The algorithm therefore only requires the radial contributions of the outgoing radiance, ignoring its directional distribution. In our method, we use a related idea.

We also pre-compute light transport in the spherical volume, but looking for the entire light field on its boundary. Since our look-up table is not limited to rays going through the sphere center, we can render particles directly by querying the outgoing intensity for any camera ray without the need for ray marching. Another conceptual difference is that the method by Ament et al. is a direct volume rendering solution. They examine volume data to fit appropriate sphere to every ray marching sample. Our method is a particle-based solution, and spheres themselves build up the cloud body. For pre-integration, we do not use Mote-Carlo simulation, but solve light transport equations with GPU-based numerical integration.

3. Mathematical Background

In this section we briefly introduce the main concepts of light transport in a participating medium. More detailed information can be found in [REK*04], [Bou08]. The three main phenomena influencing the light propagating through a participating medium are scattering, absorption and emission characterized by the scattering, absorption and emission coefficients β_{Sc}, β_{Ab} and β_{Em} correspondingly. The extinction coefficient $\beta_{Ex} = \beta_{Sc} + \beta_{Ab}$ shows the energy loss due to both out-scattering and absorption. Clouds are a purely scattering medium, i.e. absorption and emission in visible wavelengths are negligible [CS92]. Hence $\beta_{Ab} = \beta_{Em} = 0$ and $\beta_{Ex} = \beta_{Sc} = \beta$.

Optical depth $\tau(\mathbf{A},\mathbf{B})$ between points $\mathbf{A}$ and $\mathbf{B}$ is the integral of the extinction coefficient over the path:

$$\tau(\mathbf{A},\mathbf{B}) = \int_{\mathbf{A}}^{\mathbf{B}} \beta(\mathbf{P}) \cdot ds, \tag{1}$$

where $\mathbf{P} = \mathbf{A} + \vec{r} \cdot s$ is the current integration point and $\vec{r} = \frac{\mathbf{B}-\mathbf{A}}{||\mathbf{B}-\mathbf{A}||}$ is the unit direction vector between integration limits. Intensity of light traveling through the medium of optical thickness τ is reduced by a factor of $e^{-\tau}$.

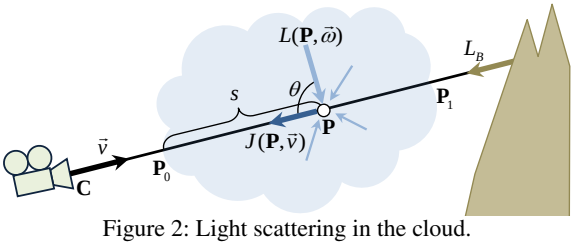

Figure 2: Light scattering in the cloud.

In-scattered light (Figure 2) at position $\mathbf{C}$ when viewing in direction $\vec{v}$ is given by the following equation:

$$L_{In}(\mathbf{C},\vec{v}) = \int_{\mathbf{P}_0}^{\mathbf{P}_1} e^{-\tau(\mathbf{P},\mathbf{P}_0)} \cdot J(\mathbf{P},\vec{v}) \cdot \beta(\mathbf{P}) \cdot ds, \tag{2}$$

where $\mathbf{P}_0$ and $\mathbf{P}_1$ are positions where the view ray enters and leaves the cloud. $J(\mathbf{P},\vec{v})$ is the total radiance of light scattered at point $\mathbf{P}$ towards the camera:

$$J(\mathbf{P},\vec{v}) = \int_{\Omega} L_{In}(\mathbf{P},\vec{\omega}) \cdot P(\theta) \cdot d\omega. \tag{3}$$

$P(\theta)$ is the phase function which describes probability of a photon being scattered from the incident direction to the outgoing direction $-\vec{v}$ (view direction $\vec{v}$ is opposite to the scattering direction, thus the "minus" sign), where θ is the angle between the two.

The main difficulty in solving eq. (2) is that L_{In} appears (implicitly through J) on both sides of the equation. This is typically solved by representing L_{In} as the infinite sum of intensities of light scattered exactly n times:

$$L_{In} = \sum_{n=0}^{\infty} L_{In}^{(n)}, \tag{4}$$

where

$$L_{In}^{(n)}(\mathbf{C},\vec{v}) = \int_{\mathbf{P}_0}^{\mathbf{P}_1} e^{-\tau(\mathbf{P},\mathbf{P}_0)} \cdot J^{(n)}(\mathbf{P},\vec{v}) \cdot \beta(\mathbf{P}) \cdot ds, \tag{5}$$

$$J^{(n)}(\mathbf{P},\vec{v}) = \int_{\Omega} L_{In}^{(n-1)}(\mathbf{P},\vec{\omega}) \cdot P(\theta) \cdot d\omega. \tag{6}$$

$L_{In}^{(0)}$ is the radiance of light scattered 0 times, which is simply external radiance attenuated by the cloud.

The phase function of cloud droplets is very complex [BH98]. In our work, we approximate it using the well-known Cornette-Shanks function [CS92]:

$$P(\theta) \approx \frac{1}{4\pi} \frac{3(1-g^2)}{2(2+g^2)} \frac{(1+\cos^2(\theta))}{(1+g^2-2g\cos(\theta))^{3/2}}. \tag{7}$$

It must be noted that our method is not limited to this simplification and it can be easily extended to use a more accurate phase function. The final radiance measured at the camera is the sum of in-scattered light L_{In} and the attenuated background radiance L_B:

$$L(\mathbf{C},\vec{v}) = L_{In}(\mathbf{C},\vec{v}) + e^{-\tau(\mathbf{P}_0,\mathbf{P}_1)} \cdot L_B. \tag{8}$$

4. Pre-computed Solution

Performance of current graphics hardware does not allow solving equations (1, 4, 5, 6) in real time. Our solution is inspired by [BN08], where the property of the spherical symmetry of the Earth's atmosphere is used to pre-compute multiple scattering in a 4D look-up table. Our idea is to select some basic relatively simple cloud particle (which we will call *reference* particle), pre-compute equations (1, 4, 5, 6) for all possible camera positions and orientations and light directions, and store the resulting data in the look-up tables. We use these look-up tables at run time to approximate lighting.

4.1. Optical Depth

Consider some inhomogeneous volume with known density distribution which will make up our reference particle. Let us assume that the camera is always located outside that volume. Our goal is to pre-compute optical depth integral (1) for all possible camera positions and orientations (Figure 3, left). In other words, eqn. (1) should be evaluated for every ray piercing the bounding sphere of that particle.

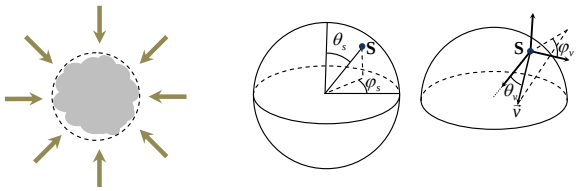

Figure 3: Reference cloud particle (left). Parameterization for the optical depth integral (middle, right).

To define the ray, we need four parameters. The first two are the azimuth ($\varphi_S \in [0, 2\pi]$) and zenith ($\theta_S \in [0, \pi]$) angles of the ray entry point **S** (Figure 3, middle). The other two define azimuth ($\varphi_v \in [0, 2\pi]$) and zenith ($\theta_v \in [0, \pi/2]$) angles of the view ray $\vec{v}$ in the tangent frame (Figure 3, right). Local z axis points inside the sphere towards its center. Notice that since we need to account for rays going inside the sphere only, maximum value for θ_v is $\pi/2$. Other parameterizations are certainly possible, but we found that this one works reasonably well.

Pre-computation then consists in going through all possible values of φ_S, θ_S, φ_v and θ_v and numerically integrating eqn. (1). We will denote the resulting solution by $\overline{T}(\varphi_S, \theta_S, \varphi_v, \theta_v)$. Extinction coefficient $\beta(\mathbf{P})$ at every ray step is obtained by combining several 3D noises. Figure 4 illustrates particles created using different methods.

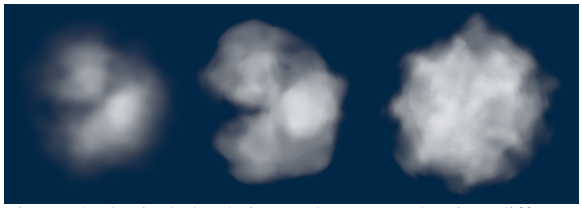

Figure 4: Optical depth integral computed using different methods: radial fall-off + 3D noise (left), 3D noise + thresholding (middle), pyroclastic style (right).

4.2. Single Scattering

Unfortunately, it is not possible to pre-compute single and multiple scattering for an inhomogeneous particle, as it requires too many parameters (at least five). However, if we assume that the particle density depends only on the distance to the center, the number of parameters can be reduced. Consider such a spherical particle illuminated by a directional light source. Our goal is to evaluate multiple scattering for every light direction, every camera position, and orientation. Due to the symmetry, we can arbitrary choose the light direction. Assume that it coincides with the positive z axis (Figure 5). Next, examine one of the rays intersecting the particle. Due to the symmetry, the light field will be symmetrical with respect to the incident light direction. Consequently, we need only zenith angle θ_S to describe the start point. We will use the same two angles in the local frame (ϕ_v and θ_v) to describe the view ray. Thus for a given density, the light field on the sphere surface can be described by three parameters. However, to evaluate eqn. (6), we will need to know the light field in the whole volume, not only on the surface. Therefore we use an intermediate 4D look-up table with the 4-th parameter being the distance r from the center. Also note that we need to cover the entire sphere of directions in local frame in this case, hence $\theta_v \in [0, \pi]$.

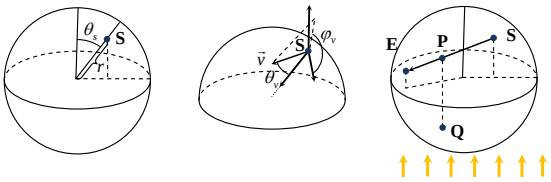

Figure 5: Pre-computing single scattering inside the spherical particle.

The solution to single scattering $\overline{L}^{(1)}_{In}(\theta_S, \phi_v, \theta_v, r)$ inside the spherical particle is then computed for each $\theta_S \in [0, \pi]$, $\phi_v \in [0, 2\pi]$, $\theta_v \in [0, \pi]$, $r \in [0, 1]$ by numerical integration of the following equation:

$$\overline{L}^{(1)}_{In} = \int_{\mathbf{S}}^{\mathbf{E}} e^{-\tau(\mathbf{P},\mathbf{S})} e^{-\tau(\mathbf{P},\mathbf{Q})} \cdot \beta(\mathbf{P}) \cdot ds, \qquad (9)$$

where **S** is the start point of the ray and **E** is the point where the ray exits the sphere (Figure 5). **Q** is the point on the surface of the sphere through which the light reaches the current integration point **P**. Note that pre-computed single scattering does not include the phase function, which is evaluated at run time to avoid precision issues caused by significant anisotropy of the function. Different methods can be used to evaluate scattering/extinction coefficient $\beta(\mathbf{P})$ as long as it depends on the distance to the sphere center only: $\beta(\mathbf{P}) = \beta(||\mathbf{P}||)$. We tried different methods and found that constant density $\beta(\mathbf{P}) = \beta$ works best.

4.3. Multiple Scattering

To find solution $\overline{L}^{M}_{In}$ to multiple scattering (2+), we iterate over scattering orders from 2 to N and for each order n perform the following steps:

1. Compute $\overline{J}^{(n)}(\theta_S, \phi_v, \theta_v, r)$ for all θ_S, ϕ_v, θ_v, r by numerically solving integral (6) for the spherical particle.
2. Compute $\overline{L}^{(n)}_{In}(\theta_S, \phi_v, \theta_v, r)$ for all θ_S, ϕ_v, θ_v, r by numerically solving integral (5).

3. Accumulate current scattering order: $\overline{L}^{M}_{In} = \overline{L}^{M}_{In} + \overline{L}^{(n)}_{In}$.

After all scattering orders are processed, we copy the radiance on the surface ($r = 1$) into the final look-up table. In contrast to optical depth, scattering in the particle is not linear with respect to density. Therefore we pre-compute single and multiple scattering as described above for a number of densities, and store the resulting data in a 4D look-up table with 4th parameter being density scale ρ.

Figure 6 demonstrates single, multiple and final lighting for the spherical particles under different lighting conditions.

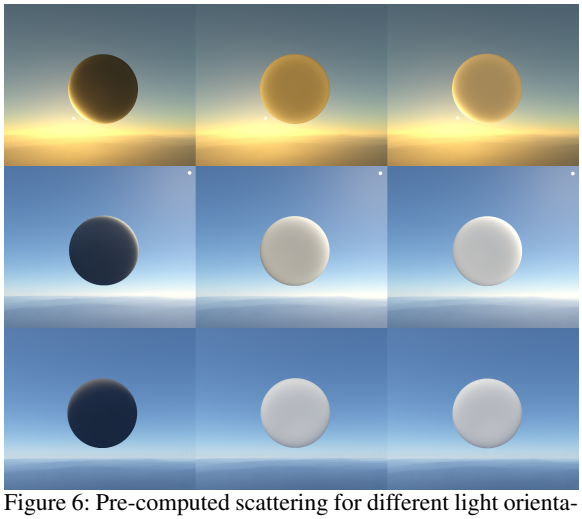

Figure 6: Pre-computed scattering for different light orientations. From left to right: single scattering only, 2+ scattering only, all terms (including ambient). The particle in the bottom row is illuminated from above.

4.4. Real-time Shading Model

Now we will describe our approximated solution to (8) using the pre-computed solutions to optical depth $\overline{T}$, single scattering $\overline{L}^{(1)}_{In}$ and multiple scattering $\overline{L}^{M}_{In}$. We will denote the corresponding solutions for a given view ray and light direction by lower case letters ($\bar{\tau}$, $\bar{l}^{(1)}$, $\bar{l}^{M}$). The first step is obtaining the optical depth $\bar{\tau}$ for the given view ray. We can get the exact value from $\overline{T}$:

$$\bar{\tau} = \overline{T}(\varphi_S, \theta_S, \varphi_v, \theta_v).$$

Before use, the value of $\bar{\tau}$ is multiplied by each particle's individual density scale $\bar{\rho}$. Since $\bar{\tau}$ is exact, transparency $e^{-\bar{\rho}\cdot\bar{\tau}}$ and attenuated background radiance are also computed exactly.

Our approximated lighting solution consists of three parts: single scattering, multiple scattering and ambient. To get an approximation to single scattering, we assume that it is equal to single scattering in the homogeneous sphere whose density scale corresponds to the optical depth of the view ray:

$$\bar{l}^{(1)} = P(\theta) \cdot \overline{L}^{(1)}(\theta_S, \phi_v, \theta_v, \bar{\rho}\cdot\bar{\tau}).$$

Note also that the phase function is not included in $\overline{L}^{(1)}$ and needs to be evaluated now. Intensity of multiple scattering is global process and more dependent on the total density scale of the whole particle. Therefore we use the following equation:

$$\bar{l}^{M} = \overline{L}^{M}(\theta_S, \phi_v, \theta_v, \bar{\rho}).$$

Single and multiple scattering are multiplied by the intensity of the attenuated sun light reaching the particle (detailed description is given in section 5.3).

Finally, ambient light $\bar{l}^{A}$ is approximated based on the observation that cloud creases are brighter than edges. To account for this, we compute the distance to the first cloud substance when pre-computing optical depth. This value is also stored in $\overline{T}$. We then use this value at run time to scale the ambient sky light intensity.

5. Implementation

5.1. Pre-computing Look-up Tables

Since current graphics hardware does not natively support 4D textures, we emulate them using 3D textures with manual interpolation for the fourth coordinate. In our implementation, optical depth integral is stored in a $64 \times 32 \times 64 \times 32$ ($N_{\varphi_S} = 64$, $N_{\theta_S} = 32$, $N_{\varphi_v} = 64$, $N_{\theta_v} = 32$) 8-bit look-up table. The table occupies 4 MB of data. Instead of storing the integral itself, we store the normalized average extinction coefficient along the ray, which lies in range $[0,1]$. Optical depth is reconstructed by multiplying that value by the ray length.

Single and multiple scattering are stored in two $32 \times 64 \times 16 \times 8$ ($N_{\theta_S} = 32$, $N_{\varphi_v} = 64$, $N_{\theta_v} = 16$, $N_\rho = 8$) 16-bit float look-up tables. Each table requires 0.5 MB of storage. We use power scale for density: $\rho = 2^s, -4 \le s < 3$. Pre-computation implements equations described in section 4. We compute up to $N = 18$ scattering orders. For single scattering we assume $g = 0.9$, for multiple scattering we use $g = 0.7$. We use $\beta = 0.07$ as scattering/extinction coefficient. All operations are implemented as pixel shaders and are executed on the GPU.

5.2. Particle Generation

We use a fully procedural method to generate particles. To account for expansive view distances, which can be covered by clouds, we exploit the camera-centered grid structure (Figure 7) inspired by the geometry clipmaps terrain rendering system [LH04]. The grid consists of a number of rectangular rings centered around the camera. Cells in the next ring are twice the size of the cells in the inner ring. Each cell contains several particle layers. The number of layers, as well as particle size and density, are determined by 2D noise. Particles in higher layers are smaller to make cumulus-like

cloud shapes with narrower tops and wider bottoms. All particles are randomly rotated and displaced to break repetitive patterns. To hide transitions between LODs, grids slightly overlap, and particles in overlapping regions are smoothly blended.

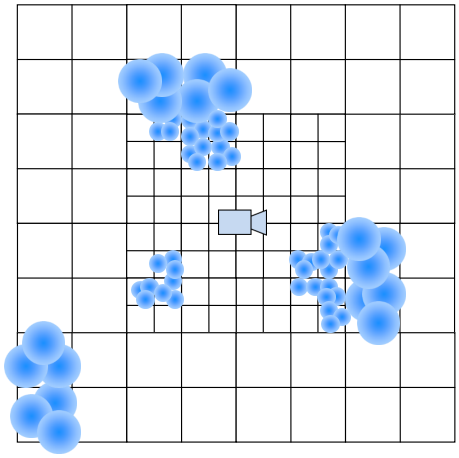

Figure 7: Camera-centered particle grid.

Particle generation is performed on the GPU and consists of the following steps:

1. Process each cell of the grid, compute cloud density in this cell, and create a list of valid non-empty cells.
2. Process the list of valid cells. For each cell, output one or more particles depending on the number of layers and cloud density in the cell.
3. Generate ordered list of particles for rendering.

The first step is implemented by the compute shader. One thread of the shader processes one cloud cell. Indices of valid cells are output to the append buffer, a DX11 object which maintains internal counter and allows creating unordered lists of elements on the GPU.

The second step is implemented by another compute shader. Since only the GPU knows the number of valid cells, we use DispatchIndirect () function to avoid CPU-GPU synchronization. The function loads its arguments from a GPU-residing buffer. We write internal counter of the append buffer to this buffer to execute the required number of GPU threads.

In the third step, we generate an ordered list of particles for rendering. For our algorithm to work properly, the particles need to be sorted from back to front. Sorting can be done on the GPU, using, for instance, bitonic sort. We use a different method. We sort all cells on the CPU, and then process the list on the GPU retaining only valid cells. At this stage append buffers cannot be used because order needs to be preserved. To guarantee original ordering, we use geometry shader and stream output stage. To improve performance, we process 32 cloud particles by every geometry shader invocation. At this stage we also test each valid particle against the view frustum planes to cull all invisible particles.

5.3. Occlusion Estimation

At this stage it is necessary to determine the amount of light reaching every particle. That is, to estimate the attenuation by other particles as the light travels to the current particle (Figure 8). We solve this problem in three steps:

1. Perform light-space tiling and construct lists of particles covering each tile.
2. Use the particle lists constructed in the first step to estimate occlusion.
3. Smooth occlusion.

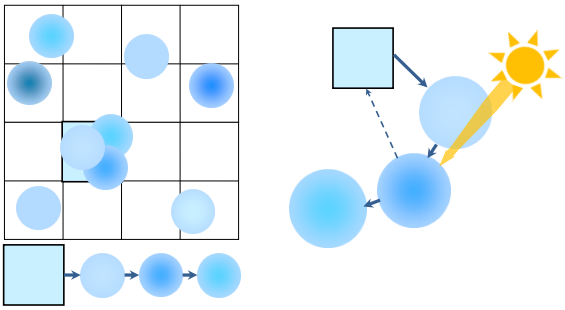

Figure 8: Light-space tile grid (left) and computing light occlusion by traversing the tile's list (right).

The first step is implemented by rasterizing the particles as seen from the light over the tile grid. Each pixel of the target viewport corresponds to one tile. Particles are extended by half the tile size to ensure conservative coverage. Typical tile grid size in our method is 64×64. The tiling information consists of two parts: an integer texture (bound as unordered access view), which has the same dimensions as the tile grid and stores indices of the first particle covering the tile, and the append buffer containing linked lists of particles. Before tiling, the first particle index texture is cleared to -1 to indicate empty lists. During the rasterization, the pixel shader allocates a new element in the append buffer, inserts the new particle to the head of the list, and updates the first particle index in the texture. Atomic operations are used to eliminate concurrent access to memory. We use pixel shader ordering, an extension introduced by Intel, to keep order (see section 5.5 for more details). Note, that particles are originally sorted from back to front, but every next particle is added to the head of the list. As a result, the final lists will be sorted from front to back.

The second step is implemented by a compute shader which processes every particle visible in the camera view frustum. The shader finds the tile which contains the center of the current particle. It then goes through the list of particles assigned to that tile. The shader intersects the ray going from the center of the current particle towards the light with

every particle in that list, and computes amount of occlusion by the intersection. For typical clouds, about 50% of energy is scattered in a 5° angle around forward direction. To account for this property, Bouthors et al. proposed embedding a strong forward peak of the phase function into the extinction [BNL06]. We exploit the same idea through the parameter ξ which we use to scale extinction coefficient β. In our experiments we used $\xi = 0.05$.

The third step smooths the occlusion and simulates the effect of light diffusion in the body of the cloud. This step is also implemented by a compute shader which processes every valid particle and performs low-pass filtering of the light occlusion. To avoid incorrect smoothing, we use particle density as weights.

5.4. Rendering

To display each volumetric particle, we rasterize its bounding box. In the pixel shader, we reconstruct the view ray and intersect it with the sphere (ellipsoid) enclosed in the box. Based on the intersection point and the ray direction, the shader loads optical depth, single and multiple scattering from the look-up tables. Then it applies the model described in section 4.4. 3D noise is used to add small details. Three buffers are generated: cloud color, transparency, and the closest distance to cloud. To improve performance, our system enables rendering the particles to low resolution buffers, which are upscaled to full resolution using edge-preserving bilateral filter.

5.5. Volume-Aware Blending

Alpha-blending is a typical way to combine two or more particles on the screen. This method does not account for intersection of volumes represented by particles and for that reason produces inaccurate results. Here we describe our new volume-aware particle blending method.

Consider two intersecting volumetric particles which have non alpha-premultiplied colors C_0 and C_1 and densities ρ_0 and ρ_1 (Figure 9).

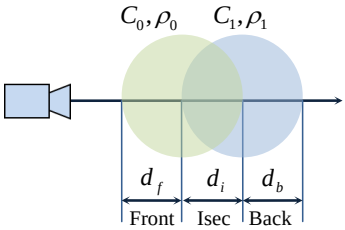

Figure 9: Intersection of two volumetric particles.

There are three regions: front, intersection, and back. For the transparency and color of the front part we can derive the following expressions:

$$T_f = e^{-\rho_0 \cdot d_f \cdot \beta} \tag{10}$$

$$C_f = C_0 \cdot (1 - T_f) \tag{11}$$

Likewise, for the back part we will obtain:

$$T_b = e^{-\rho_1 \cdot d_b \cdot \beta} \tag{12}$$

$$C_b = C_1 \cdot (1 - T_b) \tag{13}$$

For the transparency and color of the intersection we propose the following expressions:

$$T_i = e^{-(\rho_0 + \rho_1) \cdot d_i \cdot \beta} \tag{14}$$

$$C_i = \frac{C_0 \cdot \rho_0 + C_1 \cdot \rho_1}{\rho_0 + \rho_1} \cdot (1 - T_i) \tag{15}$$

Eqn. (15) for the color of the intersection produces density-weighted average color, which is reasonable. If both particles have the same densities, the resulting color will be average of the two. If the density of one particle is significantly larger, its color will govern the resulting color.

The final transparency and alpha-premultiplied color can then be computed as follows:

$$T = T_f \cdot T_i \cdot T_b \tag{16}$$

$$C = C_f + T_f \cdot C_i + T_f \cdot T_i \cdot C_b \tag{17}$$

There is no programmable blending unit on the current generation graphics hardware, so the blending according to the above equations cannot be implemented. Pixel shaders are allowed to read and write to arbitrary memory locations, however there is no way to make read-modify-write operations atomic (al least without significant performance impact). Intel recently introduced hardware extension which solves the above problem. It is called pixel shader ordering and is available through both DirectX and OpenGL APIs. The extension guarantees the following two conditions:

- All read-modify-write operations from different pixel shader instances, *which map to the same pixel*, are performed atomically.
- Pixel shader instances are executed in the same order in which primitives were submitted for rasterization.

We use this extension to implement volume-aware blending. We use the special buffer to keep track of the continuous particle volume closest to the camera, for each pixel on the screen. The following information is stored: color C, density ρ, minimum and maximum distance along the view ray d_{min} and d_{max}. The representation is initially empty. When every new particle is rasterized, its volume extent is tested against the current representation stored in the buffer. Two cases are possible here:

1. The particle does not overlap the current volume. In this case, we blend the color of the farthest of the two into the back buffer. If the new particle is closer to the camera, it replaces the current volume representation, which is preserved otherwise.

2. The particle intersects the current volume. In this case, we blend the color C_b of the back part of the two particles into the back buffer using transparency T_b. The remaining parts (front and intersection) are merged and the merged color, density and the combined extents replace the current volumetric representation (Figure 10).

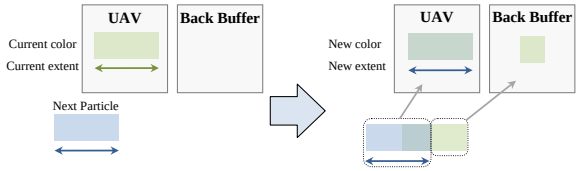

Figure 10: Updating UAV and back buffer during the volume-aware blending.

Note that for the algorithm to work properly, the particles must be processed from back to front, which is guaranteed by sorting (section 5.2).

5.6. Integration with the Atmospheric Scattering

To render the Earth's atmosphere, we adopted the technique described in [Yus14]. This technique solves the airlight integral for samples distributed along the epipolar lines on the screen, and then performs transformation from epipolar to screen space using bilateral filter. To account for occlusion by the clouds, we use the screen-size buffers generated at the particle rendering stage (section 5.4): cloud transparency, color, and minimal distance to cloud. During the ray marching, we check if the marched distance is greater than the distance to the cloud. If so, we modulate the color of each next sample by the cloud transparency (Figure 11, left). Distance to the cloud is also used to attenuate the cloud color by the atmosphere extinction.

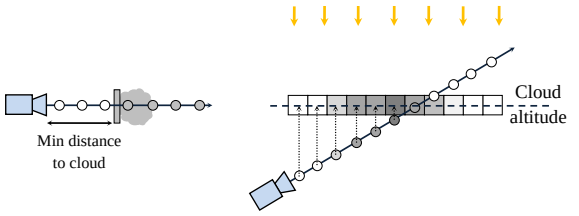

Figure 11: Integration of clouds with the atmospheric light scattering: attenuation along the view ray (left) and sun light attenuation (right).

To create the effect of shafts of light, we use the light space cloud transparency buffer which has the same structure and uses the same matrices as cascaded shadow map. At each ray marching step we check if the current position on the ray is below the clouds (clouds are assumed to have fixed altitude). If so, we load the cloud transparency from the buffer to account for the sun light occlusion (Figure 11, right).

6. Results and Discussion

We implemented the described algorithm in C++ using Direct3D11 API. The full source code is available on GitHub at https://github.com/GameTechDev/CloudySky. To explore the performance of our system, we used two test platforms. The first platform was a desktop workstation powered by Intel Core i7 CPU and NVIDIA GeForce GTX 680 GPU (195W TDP). The second test platform was an Ultrabook powered by Intel Core i5 CPU integrated with Intel HD Graphics 5200 GPU (47W TDP shared between both). All images on the first test platform were rendered in full HD resolution (1920×1080). On the second platform, the resolution was set to 1280×720. Note that our high-end platform does not support pixel shader ordering, so volume-aware blending was disabled. Also note that recently Microsoft has announced that it is going to include this feature into the next versions of Direct3D.

Creating look-up tables is performed only once, and the results are stored on the disk. On our first test platform, pre-computing optical depth integral requires 0.4 s and creating scattering look-up table takes 76 s. On our second platform, optical depth integral evaluation requires 1.1 s and pre-computing scattering takes 5 min 16 s. For comparison, CPU-based Monte-Carlo pre-integration in [ASW13] takes 2-20 hours. The optical depth can be easily updated at run time, which can be useful for switching between different types of clouds.

We evaluated run-time performance on both test platforms using three quality settings: low (ring dimension $\times$ number of rings $\times$ number of layers = $104^2 \times 4 \times 2$), medium ($136^2 \times 4 \times 4$), and high ($184^2 \times 4 \times 6$). Results are summarized in Table 1. The columns show the times in ms required to render particles, compute atmospheric effects, and perform other steps (particle generation, lighting, visibility computation, streaming). The particles were rendered in half resolution. Figure 12 shows the images rendered using these quality settings.

Plat.	Qual.	Partcl.	Atm. sctr.	Other	Total
I	Low	3.1	1.6	0.9	5.6
I	Med	6.3	1.6	1.6	9.5
I	High	10.5	1.5	3.4	15.4
II	Low	9.1	4.1	2.9	16.1
II	Med	16.8	4.2	4.0	25.0
II	High	26.2	4.2	6.1	36.5

Table 1: Run-time performance of our method (times in ms)

By changing resolution of the particle grid, the number of rings and the number of layers, our method can scale from high quality to high performance mode. Rendering larger number of small particles results in high-quality images (Figure 14). Even on low-power platform, our method is able to render convincing clouds at more than 25 fps.

Since our technique is particle-based, it provides efficient

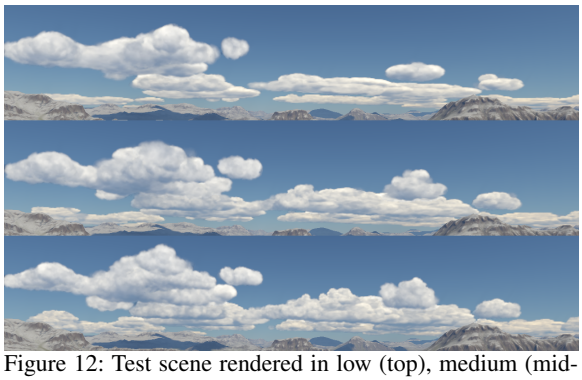

Figure 12: Test scene rendered in low (top), medium (middle) and high quality (bottom).

control over the cloud shape and locations. We used a procedural method to generate particles; however the clouds can be modeled manually by placing particles at the required positions to obtain the desired look and feel. Physics simulation e.g. [Har03], [DKY*00] or any other method can be used as well.

Similar to all particle-based methods, the performance of our algorithm is primarily determined by the number of particles rendered on the screen (see Table 1). There are many ways to improve rendering speed of such techniques. For instance, ring of impostors can be used to accelerate rendering of distant clouds in the same manner it is done in [HL01] and [Wan04].

Figure 13 demonstrates the effect of volume aware-blending. For the sake of clarity, spherical particles are used. Note that the volume-aware blending not only merges particles (even when their opacities are close to 1 as in Figure 13), but also reduces temporal artifacts caused by instant changes of particle order.

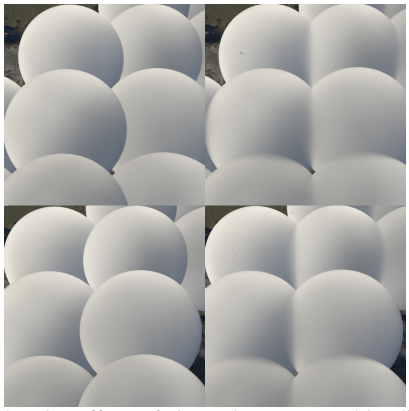

Figure 13: The effect of the volume-aware blending. Left column: alpha-blending (particle opacity is close to 1). Right column: volume-aware blending. Top row: initial camera position. Bottom row: camera moved slightly to the right causing particle reordering.

Other solutions exist that eliminate popping artifacts in volume rendering, such as the sheet buffer-based splatting by Mueller and Crawfis [MC98]. Their method however is most closely related to slicing techniques as volume data is splatted into the camera-facing slab which is advanced through the medium step-by-step. Consequently, the method cannot be used for direct rendering of intersecting particles.

6.1. Limitations

It must be noted that our method is physics-*based*, not physics-*accurate*. Though we accurately pre-compute scattering inside the reference particle, we still have to make a number of assumptions to use the data at run time. We assume, for instance, that each particle is evenly illuminated by a directional light source, which is not entirely true. This problem can be mitigated by computing attenuated light for a number of points on the particle surface. We also use phenomenological approach to account for ambient light.

Despite the fact that our volume-aware blending produces much better visual results compared to alpha-blending, it has a number of limitations. First, particles must be sorted, which is sometimes not desired. Second, only the intersection of two particles is processed precisely. Three and more overlapping particles are handled approximately. Nonetheless visual results are still good in this case. But the main problem is that the sorting is done at particle level. Consequently, there is no guarantee that volumetric fragments are always sorted from back to front. If the order is broken for two particles, the method works fine. However, if three or more fragments go in reverse order, artifacts arise, because the fragments are blended into the back buffer in wrong order. The solution to this problem requires using less dense particle sets where multiple intersections are less possible.

7. Conclusions and Future Work

We presented a new method for high-performance rendering of realistic cumulus clouds. Our method pre-computes optical depth, single and multiple scattering inside the reference particle and stores the information in the look-up tables. The tables are used at run time to approximate light transport in the cloud. Our method uses volume-aware blending to accurately mix intersecting particles. Nested grid structure is exploited to control the level of detail and enable rendering of large outdoor scenes with clouds. Despite some limitations, we believe that our idea of using pre-computed solution to shade the clouds is promising and can be further developed in future research.

In our future work, we are going to improve the performance of the algorithm. A possible way to achieve this is using impostors for distant clouds. Our lighting model can also be elaborated by modeling energy exchange between particles. Adding micro details e.g. by using additional 3D noise is another area for improvement.

Figure 14: Some high-quality images generated by our algorithm.

References

[ASW13] AMENT M., SADLO F., WEISKOPF D.: Ambient volume scattering. *IEEE Trans. Vis. Comput. Graph. 19*, 12 (2013), 2936–2945. 2, 8

[BH98] BOHREN C., HUFFMAN D. R.: *Absorption and Scattering of Light by Small Particles*. Wiley Science Paperback Series, 1998. 3

[BN08] BRUNETON E., NEYRET F.: Precomputed atmospheric scattering. *Comput. Graph. Forum 27*, 4 (June 2008), 1079–1086. Special Issue: Proceedings of the 19th Eurographics Symposium on Rendering 2008. 3

[BNL06] BOUTHORS A., NEYRET F., LEFEBVRE S.: Real-time realistic illumination and shading of stratiform clouds. In *Proceedings of the Second Eurographics Conference on Natural Phenomena* (Aire-la-Ville, Switzerland, Switzerland, 2006), NPH'06, Eurographics Association, pp. 41–50. 2, 7

[BNM*08] BOUTHORS A., NEYRET F., MAX N., BRUNETON E., CRASSIN C.: Interactive multiple anisotropic scattering in clouds. In *SI3D* (2008), Haines E., McGuire M., (Eds.), ACM, pp. 173–182. 2

[Bou08] BOUTHORS A.: *Real-time realistic rendering of clouds*. Phd thesis, Université Joseph Fourier, June 2008. 3

[CS92] CORNETTE W., SHANKS J.: Physical reasonable analytic expression for the single-scattering phase function. *Applied Optics 31*, 16 (1992), 3152–3160. 3

[DKY*00] DOBASHI Y., KANEDA K., YAMASHITA H., OKITA T., NISHITA T.: A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 19–28. 2, 9

[Ebe97] EBERT D. S.: Volumetric modeling with implicit functions: A cloud is born. In *ACM SIGGRAPH 97 Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH '97* (New York, NY, USA, 1997), SIGGRAPH '97, ACM, pp. 147–. 2

[ES00] ELINAS P., STUERZLINGER W.: Real-time rendering of 3d clouds. *J. Graphics, GPU, Game Tools 5*, 4 (2000), 33–45. 2

[Gar85] GARDNER G. Y.: Visual simulation of clouds. *SIGGRAPH Comput. Graph. 19*, 3 (July 1985), 297–304. 2

[Har03] HARRIS M. J.: *Real-time Cloud Simulation and Rendering*. PhD thesis, 2003. 9

[HH12] HUFNAGEL R., HELD M.: Star: A survey of cloud lighting and rendering techniques. *Journal of WSCG 20*, 3 (2012), 205–216. 2

[HL01] HARRIS M. J., LASTRA A.: Real-time cloud rendering. *Comput. Graph. Forum 20*, 3 (2001), 76–85. 2, 9

[JC98] JENSEN H. W., CHRISTENSEN P. H.: Efficient simulation of light transport in scences with participating media using photon maps. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 311–320. 2

[KHSM03] KNISS J., HANSEN C., SHIRLEY P., MCPHERSON A.: A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics 9* (2003), 150–162. 2

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 769–776. 5

[MC98] MUELLER K., CRAWFIS R.: Eliminating popping artifacts in sheet buffer-based splatting. In *Proceedings of the Conference on Visualization '98* (Los Alamitos, CA, USA, 1998), VIS '98, IEEE Computer Society Press, pp. 239–245. 9

[MDN04] MIYAZAKI R., DOBASHI Y., NISHITA T.: A fast rendering method of clouds using shadow-view slices. In *Proc. CGIM* (2004), pp. 93–98. 2

[NDN96] NISHITA T., DOBASHI Y., NAKAMAE E.: Display of clouds taking into account multiple anisotropic scattering and sky light. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 379–386. 2

[REK*04] RILEY K., EBERT D. S., KRAUS M., TESSENDORF J., HANSEN C.: Efficient rendering of atmospheric phenomena. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2004), EGSR'04, Eurographics Association, pp. 375–386. 2, 3

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

[SSEH03] SCHPOK J., SIMONS J., EBERT D. S., HANSEN C.: A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 160–166. 2

[Wan04] WANG N.: Realistic and fast cloud rendering. *J. Graphics, GPU, and Game Tools 9*, 3 (2004), 21–40. 2, 9

[Yus14] YUSOV E.: High performance outdoor light scattering using epipolar sampling. In *GPU Pro 5*, Engel W., (Ed.). A K Peters, 2014, pp. 101–126. 8