# A Survey of Cloud Lighting and Rendering Techniques

Roland Hufnagel
Univ. Salzburg, Salzburg, Austria
rhufna@cosy.sbg.ac.at

Martin Held
Univ. Salzburg, Salzburg, Austria
held@cosy.sbg.ac.at

## ABSTRACT

The rendering of participating media still forms a big challenge for computer graphics. This remark is particularly true for real-world clouds with their inhomogeneous density distributions, large range of spatial scales and different forms of appearance. We survey techniques for cloud visualization and classify them relative to the type of volume representation, lighting and rendering technique used. We also discuss global illumination techniques applicable to the generation of the optical effects observed in real-world cloud scenes.

## Keywords
cloud rendering, cloud lighting, participating media, global illumination, real-world clouds

## 1 INTRODUCTION

We review recent developments in the rendering of participating media for cloud visualization. An excellent survey on participating media rendering was presented by Cerezo et al. [5] a few years ago. We build upon their survey and present only recently published techniques, with a focus on the rendering of real-world cloud scenes. In addition, we discuss global illumination techniques for modeling cloud-to-cloud shadows or inter-reflection.

We start with explaining real-world cloud phenomena, state the graphics challenges caused by them, and move on to optical models for participating media. In the following sections we categorize the state-of-the-art according to three aspects: The representation of clouds (Section 2), rendering techniques (Section 3) and lighting techniques (Section 4).

### 1.1 Cloud Phenomenology

Clouds exhibit a huge variety of types, differing according to the following aspects.

**Size:** Clouds reside in the troposphere, which is the layer above the Earth's surface reaching up to heights of 9–22 km. In the mid-latitudes clouds show a maximum vertical extension of 12–15 km. Their horizontal extension reaches from a few hundred meters (Fig. 1.7) to connected cloud systems spanning thousands of kilometers (Figs. 1.11 and 1.12).

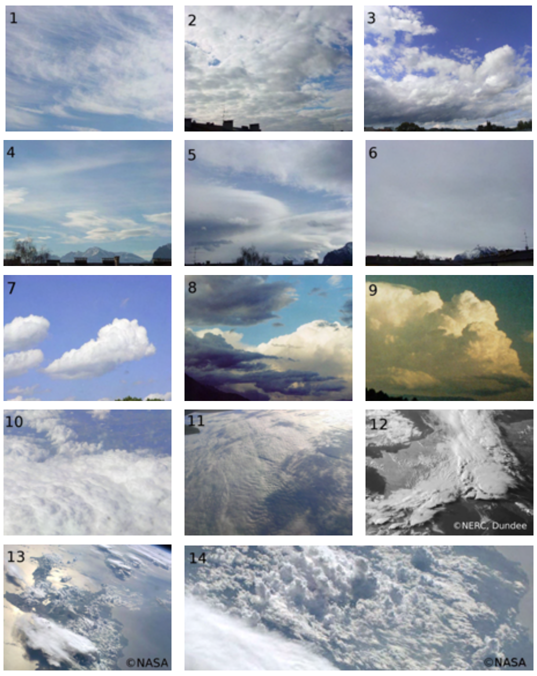

Figure 1: Clouds as seen from the ground: broken cloud layers with fractal cloud patches (top row), clouds with diffuse boundaries (second row), dense cloud volumes with surface-like boundaries (third row); and clouds viewed from a plane and from space (last two rows), where (14) shows a zoom into the central region of (13).

**Geometry:** In relation to the Earth's radius (6371 km) the troposphere represents a shallow spherical shell. The curvature of the Earth produces the horizon and is directly visible when viewing clouds from space.

Clouds often develop at certain heights and form layers. These either consist of cloud patches (Fig. 1.1 to 1.5) or create overcast cloud sheets (Fig. 1.6 or 1.11).

Local upward motions emerging from the ground (convective plumes) create clouds with a sharp cloud base and a cauliflower-like structure above, so-called Cumulus (Figs. 1.7 to 1.9). Their vertical extension reaches up to several kilometers (Fig. 1.9).

For low clouds also mixed forms of convective and layered clouds exist, where the convective plumes are vertically bound and form a layer (Fig. 1.3).

Clouds formed in connection with frontal systems usually show a large vertical extension and can span thousands of kilometers (Fig. 1.12).

**Boundary Appearance:** The appearance of clouds mainly depends on the cloud volume's constituents: droplets of different size or ice particles. While large water droplets produce a determined, surface-like boundary (Figs. 1.7 to 1.9), smaller droplets create a diffuse or fractal boundary (Figs. 1.2 to 1.6). Ice particles often form hair-like or fiber-like clouds, so-called Cirrus (Fig. 1.1), or diffuse clouds, the anvil-like tops of convective clouds (Fig. 1.13).

The appearance of a cloud is strongly influenced by the distance to its observer: While distant clouds often show a distinct surface and sharp contours (Figs. 1.12 and 1.13), a closer look reveals diffuse or fractal structures (Fig. 1.14).

**Optical Phenomena:** Clouds consist of water droplets and ice crystals which scatter light mostly independent of wavelength. Clouds are therefore basically white. Spectral colors appear only at certain angular constellations but generally do not influence their overall appearance. However, several optical phenomena determine their characteristic, natural appearance:

**Self-Shadowing:** The attenuation of light within a cloud creates gray tones and is proportional to the optical depth of the volume. The self-shadowing provides the cue to perceive clouds as volumetric objects.

**Multiple scattering** slightly attenuates the effect of self-shadowing by distributing light within the cloud volume in a diffusion-like process; see e.g. [4, 27, 33].

**Inter-Cloud Shadows:** Clouds cast shadows onto other clouds, like in Fig. 1.11, where a high cloud layer on the right-hand side shadows a low cloud layer.

**The Earth's Shadow:** Clouds can be shadowed by the Earth; see Fig. 1.8 showing an evening scene, where the low clouds lie in the shadow of a mountain range.

**Indirect Illumination:** Light inter-reflection between different clouds or between different parts of the same cloud brighten those regions, as, e.g., in Fig. 1.9, where the cloud seems to gloom from the inside.

**Light Traps:** Light inter-reflection at a smaller scale occurs on determined cloud surfaces (Neyret [29]) and lets concavities appear brighter (Figs. 1.7 and 1.10).

**Corona:** The corona effect occurs when the cloud is lit from behind. The strong forward scattering at the boundary produces a bright silhouette (silver-lining).

**Atmospheric Scattering:** Clouds often appear in vivid colors. This is caused by the scattering of light outside the cloud volume on air molecules and aerosols. Clouds are therefore often lit by yellowish to reddish sunlight (Fig. 1.9). Different paths of light in the atmosphere let the high clouds in Fig. 1.14 appear bright white and the low clouds yellowish. Atmospheric scattering also creates blue skylight which in some situations represents the main source of lighting (Fig. 1.8).

**Ground Inter-Reflection:** For low-level clouds the inter-reflection with the ground creates subtle tones depending on the type of the ground; see e.g. [3].

### *1.1.1 Summary*

From a computer graphics point of view we identify the following volume properties, in addition to the form of the clouds as created by cloud modeling techniques, which are out of the scope of this survey: **thin** clouds that show no self-shadowing; cloud patches that represent a mixture of slightly dense cores and optically thin boundaries, usually forming horizontally extensive **layered** clouds; and **dense** cloud volumes of different sizes and extensions with a sharp or surface-like boundary. The boundary of clouds exhibits either a **fractal**, **diffuse** or **sharp** appearance.

## 1.2 Computer Graphics Challenges

A realistic visualization of clouds requires to tackle the following challenges:

**Heterogeneity:** Real-world cloud scenes typically consist of a very heterogeneous collection of clouds with different appearances, sizes and forms. Different cloud types require different volume representations, lighting and rendering techniques.

**Atmospheric Scattering:** For creating a cloud's natural appearance the lighting model employed has to reproduce its typical optical phenomena (see Sec. 1.1), including atmospheric scattering which is the main source for color in the sky. This requires the inclusion of atmospheric models (which are not discussed in this survey) in the lighting process of clouds.

**Curved volume:** The spherical atmosphere makes it difficult to take advantage of axis-aligned volumes if clouds are viewed on a large or even global scale, as in Figs. 1.12 or 1.13.

**Huge domain:** The sheer size of the volume of real-world cloud scene, especially when viewed from above, like in Figs 1.11 to 1.13, requires sophisticated and efficient lighting and rendering techniques.

## 1.3 Participating Media

A cloud volume constitutes a participating medium exhibiting light attenuation and scattering. This section uses the terminology of [5] to provide a short introduction to the radiometry of participating media.

While light in vacuum travels along straight lines, this is not the case for participating media. Here photons interact with the medium by being scattered or absorbed. From a macroscopic point of view light spreads in participating media, gets blurred and attenuated, similar to heat diffusion in matter.

Participating media are characterized by a particle density $\rho$, an absorption coefficient $\kappa_a$, a scattering coefficient $\kappa_s$, and a phase function $p(\vec{\omega},\vec{\omega}')$ which describes the distribution of light after scattering.

**Absorption** is the process where radiation is transformed to heat. The attenuation of a ray of light with radiance $L$ and direction $\vec{\omega}$ at position $x$ (within an infinitesimal ray segment) is described by

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = -\kappa_a(x)\,L(x,\vec{\omega}).$$

In the atmosphere absorption is mainly due to water vapor and aerosols. Cloud droplets or ice crystals show little absorption which means that the light distribution in clouds is dominated by scattering.

**Scattering** is the process where radiance is absorbed and re-emitted into other directions. *Out-scattering* refers to the attenuation of radiance along direction $\vec{\omega}$ due to scattering into other directions:

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = -\kappa_s(x)\,L(x,\vec{\omega}).$$

*In-scattering* refers to the scattering of light into the direction $\vec{\omega}$ from all directions (integrated over the sphere) at a point $x$:

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = \frac{\kappa_s(x)}{4\pi}\int_{4\pi} p(\vec{\omega}',\vec{\omega})\,L(x,\vec{\omega}')\,d\omega',$$

*Extinction* is the net effect of light attenuation due to absorption and out-scattering described by the extinction coefficient $\kappa_t = \kappa_a + \kappa_s$.

**Emission** contributes light to a ray:

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = \kappa_a(x)\,L_e(x,\vec{\omega}).$$

It is usually not relevant for cloud rendering since clouds do not emit light. (An exception is lightning inside a cloud.)

**Radiative Transfer Equation (RTE):** Putting all terms together yields the RTE which describes the change of radiance within a participating medium at a point $x$:

$$\begin{aligned}(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) \;=\;& \kappa_a(x)\,L_e(x,\vec{\omega}) \;+\\ & \frac{\kappa_s(x)}{4\pi}\int_{4\pi} p(\vec{\omega}',\vec{\omega})\,L(x,\vec{\omega}')\,d\omega' \;-\\ & \kappa_a(x)\,L(x,\vec{\omega}) \;-\; \kappa_s(x)\,L(x,\vec{\omega}).\end{aligned}$$

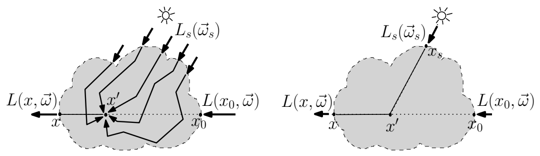

Figure 2: Multiple scattering (left), and the single scattering approximation (right).

By using the single scattering albedo $\Omega = \kappa_s/\kappa_t$ and noting that $\kappa_a$ can be expressed as $\kappa_a = \kappa_t\,(1-\Omega)$, we can re-write the RTE as

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) \;=\; \kappa_t(x)\,J(x,\vec{\omega}) \;-\; \kappa_t(x)\,L(x,\vec{\omega}), \qquad (1)$$

with the source radiance $J$:

$$\begin{aligned}J(x,\vec{\omega}) \;=\;& (1-\Omega(x))\,L_e \;+\\ & \frac{\Omega(x)}{4\pi}\int_{4\pi} p(\vec{\omega}',\vec{\omega})\,L(x,\vec{\omega}')\,d\omega'.\end{aligned}$$

The source radiance describes all contributions of radiance to a ray $(x,\vec{\omega})$ at a point $x$ inside the medium. In high-albedo media, like clouds, the source term is mainly due to in-scattering, while the extinction is dominated by out-scattering: $\kappa_t \approx \kappa_s$.

Integrating the RTE along a ray from $x_0$ to $x$ yields the radiance reaching point $x$ from direction $-\vec{\omega}$ (see Fig. 2, left):

$$L(x,\vec{\omega}) = T(x,x_0)L(x_0,\vec{\omega}) + \int_{x_0}^{x} T(x,x')\,\kappa_t(x')\,J(x')\,dx', \qquad (2)$$

with the transmittance $T(x_1,x_2) = \exp(-\int_{x_1}^{x_2}\kappa_t(x)\,dx)$. The boundary condition of the integral is $L(x_0,\vec{\omega})$, representing the light coming from the background, an environment map, or from scene objects.

Note that the coefficients $\kappa_*$ depend on the wavelength. Therefore, three versions of Eqn. 2 have to be solved with appropriate coefficients $\kappa_{*,\lambda}$ for each wavelength corresponding to the RGB color components.

**Single Scattering Approximation:** A difficulty in solving Eqn. 2 is that $L$ appears (implicitly through $J$) on both sides of the equation. A common approximative solution is to account only for a certain number of scattering events and apply extinction on the paths in between. Considering only the first order of scattering yields the single scattering approximation: The source radiance $J_{SS}$ is given by the light from the light source $L_s$ attenuated on its way between $x_s$ and $x'$, see Fig. 2 (right), thus eliminating $L$:

$$J_{SS}(x',\vec{\omega}) = \Omega(x')\,T(x',x_s)\,p(x',\vec{\omega}_s,\vec{\omega})\,L_s(x_s,\vec{\omega}_s).$$

The single scattering approximation simulates the self-shadowing of a volume. Higher order scattering accounts for the "more diffuse" distribution of light within the volume.

**Phase Function:** A scattering phase function is a probabilistic description of the directional distribution of scattered light. Generally it depends on the wavelength, and the form and size of the particles in a medium. Usually phase functions show a symmetry according to the incident light direction, which reduces it to a function of the angle $\theta$ between incident and exitant light $p(\theta)$.

Often approximations for certain types of scattering are used, like the *Henyey-Greenstein* or the *Schlick* function. However, as noted in [4], those functions cannot model visual effects that depend on small angular variations, like glories or fog-bows. See [4] and its references for plots and tabular listings of phase functions.

## 2 CLOUD REPRESENTATIONS

A cloud representation specifies the spatial distribution, overall structure, form, and boundary appearance of clouds in a cloud scene.

## 2.1 Hierarchical Space Subdivision

### *2.1.1 Voxel Octrees*

Voxel octrees are a hierarchical data structure built upon a regular grid by collapsing the grid cells of a $2 \times 2 \times 2$ cube (children) to a single voxel (parent).

**Sparse voxel octrees** reduce the tree size by accounting for visibility and LOD: Interior voxels are removed, yielding a hull- or shell-like volume representation (see Fig. 3, middle). Also hidden voxels (relative to the current viewing point) are removed (see Fig. 3, right). Additionally the LOD resolution can be limited according to the screen resolution (view-dependent LOD). These techniques require an adaptive octree representation accompanied with an update strategy.

Crassin et al. [7], and similarly Gobetti et al. [13], propose a dynamic octree data structure in combination with a local regular grid representation. Each node of the octree is associated with a *brick*, a regular $32^3$ voxel grid, which represents a filtered version of the volume enclosed by its child nodes. The bricks are stored in a *brick pool* of a fixed size. Bricks are referenced by pointers stored in the nodes of the octree (see Fig. 4). The octree nodes themselves are stored in a pool as well (*node pool*). During the visualization brick and node data is loaded on demand and the pools are managed by updating least recently used data.

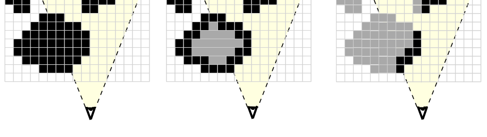

Figure 3: Voxel volume (left), shell-like boundary voxels (middle), culling invisible voxels (right).

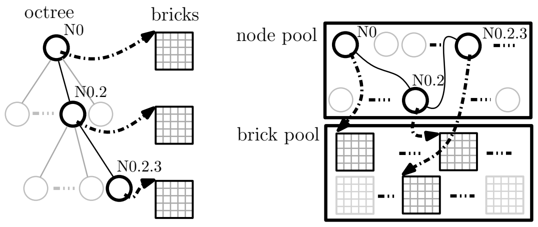

Figure 4: Sketch of the GigaVoxel data structure [7].

Laine and Karras [25] store an octree data structure in a single pool. The data is divided into blocks which represent contiguous areas of memory allowing local addressing and taking advantage of fast memory access operations on the GPU (caching). The data structure is designed to compactly store mesh-based scenes but their open-source implementation could probably be adapted to represent cloud volumes.

Miller et al. [28] use a grid representation on the coarse scale and nest octrees within those cells. This provides fast grid marching on a large scale, and adaptive sampling on the small scale. A fixed data structure, resembling 4-level octrees, allows to directly access the octree's nodes without requiring pointers. However, their current implementation assumes that the whole scene fits into the memory of the graphics device, which limits the model size. A streaming-based data structure for dynamic volumes was announced as future work.

The real-time voxelization of scene geometry, as proposed by Forester et al. [12], allows to transform rasterizable geometry to an octree representation on-the-fly on the GPU, and thus to apply voxel-based lighting and rendering to a surface-based geometry.

Octrees are a flexible data structure, generally capable of representing all types of clouds and supporting efficient lighting and rendering techniques. However, since octrees usually take advantage of axis-aligned grids, they are not directly applicable in an efficient way to large-scale cloud scenes, but would have to be nested in the spherical shell or used with a ray caster that takes into account the curvature of the Earth.

### *2.1.2 Binary Space Partitioning (BSP)*

BSP, e.g., in form of kd-trees, recursively subdivides the space into half-spaces, concentrating at regions with high geometric detail and removing empty space. BSP could be used for cloud volume representation as well.

### *2.1.3 Bounding Volume Hierarchies (BVH)*

BVH enclose scene geometry by bounding planes. Recently, BVH were used for structuring particle systems [14], which could also be employed for cloud volumes.

## 2.2 Implicit Representations

A common way to model and represent a cloud's density field is the use of procedural methods. While the

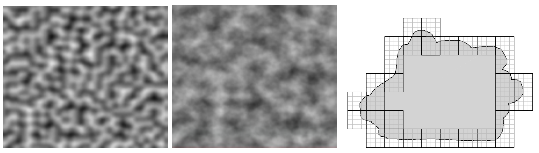
Figure 5: Perlin noise [31] and "fractal sum" [11] functions (left). A hypertexture applied to a surface (right).

overall structure is usually specified by simple geometric primitives, like spheres or ellipsoids, the internal, high-resolution structure is modeled by a function.

Perlin and Hoffert [31] introduce space-filling shapes, based on procedural functions, so-called *hypertextures*. Ebert et al. [11] propose many additional noise functions (see Fig. 5, left), and create various natural patterns applicable also for clouds.

The sampling of implicitly defined densities can become expensive and harm rendering performance. Schpok et al. [36] propose to evaluate the procedural functions on-the-fly on the GPU during the rendering by using a fragment shader program and a 3D texture containing Perlin noise.

Kniss et al. [22] use procedural functions for geometric distortion which adds a fractal appearance to regular shapes by changing the vertex positions of the geometry rendered. They apply this distortion during the rendering process by using vertex shader programs.

Bouthors et al. [4] use hypertextures in combination with surface-bounding volumes for creating a fractal boundary appearance; see Fig. 5, right.

Implicitly specifying a volume via procedural functions is a compact volume representation. The computational cost can be mitigated by employing parallel processing on the GPU and taking advantage of hardware-supported tri-linear interpolation of 3D textures. Procedural techniques are perfect for clouds with a fractal boundary appearance. However, they only provide the fine-scale volume structure while the overall cloud shape has to be modeled by other techniques.

## 2.3 Particle Systems

The volume is represented by a set of particles with a pre-defined volume. Usually spherical particles with a radial density function are used.

Nishita et al. [30] promote the use of particles with a Gaussian density distribution, so-called *metaballs*. While the metaballs in [30] are used only to create a volume density distribution, Dobashi et al. [10] directly light and render the metaballs and visualize medium-size cloud scenes of Cumulus-like clouds with a diffuse boundary appearance.

Bouthors and Neyret [2] use particles to create a shell-like volume representation for Cumulus-like clouds. Their algorithm iteratively places particles at the interface of a cloud volume, with smaller particles being placed upon the interface of larger particles. This creates a hierarchy of particles with decreasing radius and specifies the cloud's surface, which can be transformed, e.g., to a triangle mesh.

Efficient transformation algorithms were developed to match the favored lighting and rendering approaches. Cha et al. [6] transform the density distribution given by a particle system to a regular grid by using the GPU, while Zhou et al. [44] propose the inverse process transforming a density field to *radial basis function* (RBF) representation. This low-resolution particle system is accompanied by a high-resolution grid, which stores deviations of the particles' density distribution from the initial density field in a so-called *residual field. Perfect spatial hashing* allows a compact storage of this field.

Particles systems are a compact volume representation and directly support many cloud modeling techniques. Spherical particles are well suited for Cumulus-like clouds or dense cloud volumes, but less appropriate for stratified cloud layers with a large horizontal extension or for thin, fiber-like clouds.

## 2.4 Surface-Bounded Volumes

The cloud volume is represented by its enclosing hull, usually given as a triangle mesh. Since no information on the internal structure is available, usually a homogeneous volume is assumed.

Bouthors et al. [4] demonstrate the use of surface-bounded volumes for visualizing single Cumulus clouds. A triangle mesh is used in combination with a hypertexture to add small-scale details at the boundaries. A sophisticated lighting model reproduces a realistic appearance (see Sec. 4.1.4).

Porumbescu et al. [32] propose *shell maps* to create a volumetric texture space on a surface. A tetrahedral mesh maps arbitrary volumetric textures to this shell.

Surface-bounded volumes are a very compact representation of cloud volumes, allowing for efficient rendering and for incorporating sophisticated lighting techniques. However, they are only applicable if a quick saturation of a ray entering the volume may be assumed. While this is valid for dense clouds it does not apply to thin or layered clouds with their optically thin boundaries. Also special rendering techniques have to be developed for allowing perspectives from within the clouds.

## 2.5 Clouds as Layers

Clouds represented by a single layer, usually rendered as a textured triangle mesh, allow fast rasterization-based rendering and are the traditional technique to present clouds, e.g., during weather presentations [24].

Bouthors et al. [3] visualize cloud layers viewed from the ground or from above. They achieve a realistic appearance of the cloud by applying a sophisticated, viewpoint-dependent lighting model.

The 2D representation is especially predestined for thin cloud sheets viewed from the ground, or for visualizing the cloud tops viewed, e.g., from the perspective of a satellite. However, this non-volumetric representation limits the possible perspectives and generally does not allow for animations, like transitions of the viewpoint from space to the ground, or cloud fly-throughs.

## 3 CLOUD RENDERING TECHNIQUES

## 3.1 Rasterization-based Rendering

### *3.1.1 Volume Slicing*

Volume slicing is a straightforward method for rendering regular grids. The slices are usually axis aligned and rendered in front-to-back order (or vice-versa), applying viewing transformations and blending. While volume slicing is not necessarily a rasterization-based method, most algorithms exploit the highly optimized texturing capabilities of the graphics hardware.

Schpok et al. [36] use volume slicing for cloud rendering and propose the use of the GPU also for adding detailed cloud geometry on-the-fly by using a fragment shader program. This reduces the amount of data which has to be transferred onto the GPU to a low-resolution version of the cloud volume. Harris et al. [15] create the density volume by a CFD simulation on the GPU for creating a 3D density and light texture which can efficiently be rendered on the GPU. Sparing the transfer of the volumetric data from the CPU, they achieve interactive frame rates for small volumes (up to $64^3$), creating soft, diffuse clouds. Hegeman et al. [17] also use 3D textures to capture the volume density and the pre-calculated source radiance, and evaluate a lighting model in a CG shader program during rendering.

Zhou et al. [44] visualize smoke represented by a low-resolution particle system accompanied by a high-resolution density grid, stored in compressed form (see Sec. 2.3). During the rendering process the lighted particles are first rasterized and converted to a 3D texture by applying parallel projection rendering along the $z$-axis to fill slices of the 3D texture. Thereby, for each slice, all particles are traversed and, in case of intersection with the slicing plane, rendered as textured quads. During this pass also the residual field is evaluated and stored in a separate 3D texture. In a second step perspective rendering is applied by slicing in back-to-front manner. Again, for each slice all particles are traversed and bounding quads are rendered triggering a fragment shader which composes the light and density information from the 3D textures. They achieve interactive frame rates under dynamic lighting conditions for moderate volume sizes which completely fit into the GPU's memory ($128^3$ in their examples), based on 0.5 to 1.5 hours of pre-processing time.

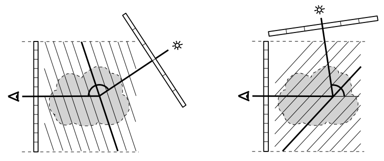
Figure 6: Volume rendering by half-angle slicing.

**Half-Angle Slicing:** Slicing the volume at planes oriented halfway between the lighting and viewing direction (or its inverse, see Fig. 6) is called half-angle slicing. It allows to combine the lighting and rendering of the volume in a single process by iterating once over all slices. During this single-volume pass two buffers are maintained and iteratively updated: one for accumulating the attenuation of radiance in the light direction, and one for accumulating the radiance for the observer (usually in the frame buffer). Due to the single pass through the volume, the lighting scheme is limited to either forward or backward scattering.

Kniss et al. [22] use half-angle slicing in combination with geometric distortion, modifying the geometry of shapes on-the-fly during rendering (see Sec. 2.2) by using vertex shader programs. Riley et al. [35] visualize thunderstorm clouds, taking into account different scattering properties of the volume.

For avoiding slice-like artifacts volume slicing techniques have to employ small slicing intervals, resulting in a large number of slices which can harm rendering performance and introduce numerical problems. Inhomogeneous volumes, like in Figs. 1.2 or 1.14, would also require a huge number of slices for appropriately capturing the volume's geometry and avoiding artifacts in animations with a moving viewpoint. Slicing-based methods therefore seem to favor volumes with soft or diffuse boundaries and, thus, are applicable to clouds with sharp boundaries only to a limited extent.

### *3.1.2 Splatting*

Splatting became the common method for rendering particle systems. Particles, which are usually specified as independent of rotation, can be rendered by using a textured quad representing the projection of the particle onto a plane, also called splat or footprint. The particles are rendered in back-to-front order, applying blending for semi-transparent volumes.

Dobashi et al. [10] use metaballs for splatting cloud volumes. Harris et al. [16] accelerate this approach by re-using impostors, representing projections of a set of particles, for several frames during fly-throughs. The use of different particle textures can enhance the clouds' appearance [42, 18].

Since usually a single color (or luminance) is assigned to each particle and the particles do not represent a distinct geometry, but a spherical, diffuse volume (due to the lack of self-shadowing within a particle), the splatting approach is limited to visualizing clouds with a soft, diffuse appearance. Clouds with a distinct surface geometry, like Cumulus clouds, cannot be reproduced realistically.

#### *3.1.3 Surface-Based Volume Rendering*

Nowadays, rendering clouds as textured ellipsoids is no more regarded as realistic. The same applies to the surface-bounded clouds of [41]. A triangle mesh is created by surface subdivision of an initial mesh, created by a marching cubes algorithm applied on weather forecast data. The rendering of the semi-transparent mesh, however, is prone to artifacts on the silhouette.

Bouthors et al. [4] resurrected surface-based cloud rendering by using fragment shader programs which calculate the color for each fragment of a triangle mesh at pixel-basis. This allows to apply a sophisticated volume lighting scheme and the sampling of a hypertexture superposed onto the surface on-the-fly for each pixel. They achieve a realistic, real-time visualization of dense clouds with fractal and sharp boundaries.

### 3.2 Ray Casting-Based Rendering

#### *3.2.1 Ray Marching*

Ray marching casts rays into the scene and accumulates the volume densities at certain intervals. For rendering participating media volumes, like clouds, the illumination values of the volume have to be evaluated, either by applying a volume lighting model on-the-fly or by retrieving the illumination from a pre-computed lighting data structure.

**Grids:** Cha et al. [6] transform a particle-based volume representation to a regular density grid for applying ray marching. The volume density is sampled within a 3D texture, combined with the illumination, stored in a separate 3D texture, and accumulated along the viewing rays. Geometric detail is added to the low-resolution volume representation on-the-fly by slightly distorting the sampling points of the ray march according to a precomputed 3D procedural noise texture. For volumes fitting into the memory of the GPU (around $256^3$ in their examples), they achieve rendering times of a few seconds (including the lighting calculation).

**BVHs:** A particle system stored within a kd-tree structure is proposed by Gourmel et al. [14] for fast ray tracing. The technique could be used for cloud rendering, e.g., by ray marching through the particle volumes and adding procedural noise as proposed in [4] or [6].

**Octrees:** The huge amount of data caused by volumetric representations can be substantially reduced by using ray-guided streaming [7, 13, 25]. The GigaVoxel algorithm [7] is based on an octree with associated bricks (Sec. 2.1.1) which are maintained in a pool and streamed on demand. The bricks at different levels of the octree represent a multi-resolution volume, similar to a mip-map texture. Sampling densities at different mip-map resolutions simulates cone tracing (see Fig. 4, right). The octree is searched in a stack-less manner, always starting the search for a sampling position at the root node. This supports the cone-based sampling of the volume since the brick values at all levels can be collected during the descent. The implementation employs shader programs on the GPU, and achieves 20–90 fps, with volumetric resolutions up to 160k.

## 4 LIGHTING TECHNIQUES

### 4.1 Participating Media Lighting

We review recent volume lighting approaches and refer to [5] for a survey of traditional, mainly off-line lighting models.

#### *4.1.1 Single-Scattering Approximation*

**Slice-Based:** Schpok et al. [36] use volume slicing for creating a low-resolution volume storing the source radiances. This so-called light volume is oriented such that light travels along one of the axes, which allows a straightforward light propagation from slice to slice. The light volume storing the source radiance is calculated on the CPU and transferred to a 3D texture on the GPU for rendering.

In the half-angle slicing approach by Kniss et al. [22, 23] the light is propagated from slice to slice through the volume by employing the GPU's texturing and blending functionalities. A coarser resolution can be used for the 2D light buffer (Fig. 6), thus saving resources and accelerating the lighting process.

**Particle-Based:** Dobashi et al. [10] use shadow casting as a single-scattering method for lighting a particle system. The scene is rendered from the perspective of the light source, using the frame buffer of the GPU as a shadow map. The particles are sorted and processed in front-to-back manner relative to the light source. For each particle the shadow value is read back from the frame buffer before proceeding to the next particle and splatting its shadow footprint. This read-back operation forms the bottleneck of the approach which limits either the model size or the degree of volumetric detail. Harris and Lastra [16] extend this approach by simulating multiple forward scattering and propose several lighting passes for accumulating light contributions from different directions including skylight.

Bernabei et al. [1] evaluate for each particle the optical depth towards the boundary of the volume for a set of directions and store it in a spherical harmonic representation. Each particle therefore provides an approximate optical depth of the volume for all directions, including the viewing and lighting directions (Fig. 8, left). The rendering process reduces to accumulating the light contributions of all particles intersecting the viewing ray, thus sparing a sorting of the particles, and provides interactive frame rates for static volumes. For the source radiance evaluation a ray marching on an intermediate voxel-based volume is used in an expensive pre-process. Zhou et al. [44] accomplish this in real-time by employing spherical harmonic exponentiation; see Sec. 4.1.6.

#### *4.1.2 Diffusion*

Light distribution as a diffusion process is a valid approximation in optically thick media, but not in inhomogeneous media or on its boundary. Therefore Max et al. [27] combine the diffusion with anisotropic scattering and account for cloudless space to reproduce, e.g., silver-lining. However, the computational cost is still high and causes rendering times of several hours.

#### *4.1.3 Path Tracing*

Path tracing (PT) applies a Monte Carlo approach to solve the rendering equation. Hundreds or even thousands of rays are shot into the scene for each pixel and traced until they reach a light source. PT produces unbiased, physically correct results but generally suffers from noise and low convergence rates. Only recent acceleration techniques allow an efficient rendering of large scenes, at least for static volumes.

In [39, 43], the volume sampling process of PT is accelerated by estimating the free path length of a medium in advance and by using this information for a sparse sampling of the volume ("*woodcock tracking*"). While Yue et al. [43] use a kd-tree for partitioning the volume, Szirmay-Kalos et al. [39] use a regular grid.

#### *4.1.4 Path Integration*

Path integration (PI) is based on the idea of following the path with the highest energy contribution and estimates the spatial and angular spreading of light along this so-called *most probable path* (MPP). PI favors high-order scattering with small scattering angles, and tends to underestimate short paths, diffusive paths and backward scattering.

The initial calculation model of Premoze et al. [33] is based on deriving a *point spread function* which describes the blurring of incident radiance. Hegeman et al. [17] proposed the evaluation of this lighting model using a shader program executed on the GPU. They visualize dynamic smoke (at resolutions up to $128^3$) within a surface-based scene at interactive frame rates.

Bouthors et al. [4] use PI in combination with pre-computed tables which are independent of the cloud volume. In an exhaustive pre-computation process the impulse response along the MPP at certain points in a slab is calculated and stored as spherical harmonics (SH) coefficients (Fig. 7, left). During the rendering process the slabs are adjusted to the cloud volume (Fig. 7, right), and used for looking up the pre-computed light attenuation values. Multiplying it with the incident radiance at the cloud surface around the so-called collector area yields the resulting illumination. The model is implemented as a shader program which evaluates the lighting model pixel-wise in real-time.

#### *4.1.5 Photon Mapping*

Photon mapping (PM) traces photons based on Monte Carlo methods through a scene or volume and stores them in a spatial data structure that allows a fast nearest-neighborhood evaluation (usually a kd-tree). However, when applied to volumes the computational and memory costs are significant.

Cha et al. [6] combine photon tracing with irradiance caching, accumulating the in-scattered radiances at voxels of a regular grid. The gathering process reduces to a simple ray marching in the light volume executed on the GPU. This provides rendering times of a few seconds for medium-size volumes.

**Progressive photon mapping** as proposed by Knaus and Zwicker [21] improves traditional static photon mapping (PM) by reducing the memory cost and can also be applied to participating media. Photons are randomly traced through the scene, but instead of storing them in a photon map, they are discarded and the radiances left by the photons are accumulated in image space. Jarosz et al. [20] combine progressive PM with an improved sampling method of photon beams.

#### *4.1.6 Mixed Lighting Models*

Lighting techniques can be combined by evaluating different components of the resulting light separately.

**Particle-Based:** Zhou et al. [44] evaluate the source radiances at the particles' centers by accumulating the light attenuation of all particles onto each particle. Thereto a convolution of the background illumination function with the light attenuation function of the particles (spherical occluders) is applied (Fig. 8,

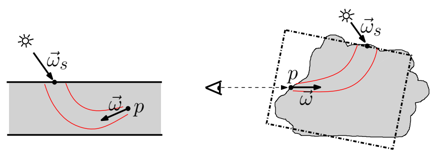

Figure 7: Pre-computed light in a slab (left), fitting slabs to a cloud surface (right).

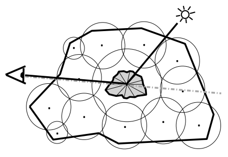
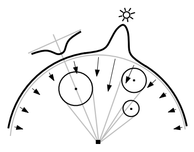
Figure 8: Pre-computed optical density at a particle center (left). Convolution of background illumination with the light attenuation by spherical occluders (right).

right). The functions are represented by low-order spherical harmonics (SH) and the convolution is done in logarithmic space where the accumulation reduces to summing SH coefficients; a technique called SH exponentiation (SHEXP), introduced by Ren et al. [34] for a shadow calculation. For multiple scattering an iterative diffusion equation solver is used on the basis of the source radiance distribution. This includes solving linear systems of dimension $n$, with $n$ being the number of particles. The approximation of the smoke density field by a small number of particles (around 600 in their examples) allows interactive frame rates under dynamic lighting conditions. However, the super-quadratic complexity in terms of the particle number prohibits the application to large volumes.

**Surface-Based:** Bouthors et al. [3] separately calculate the single and multiple-scattering components of light at the surface of a cloud layer. The single-scattering model reproduces silver lining at the silhouettes of clouds and back-scattering effects (glories and fog-bows). Multiple-scattering is evaluated by using a BRDF function, pre-calculated by means of path tracing. Inter-reflection of the cloud layer with the ground and sky is taken into account by a radiosity model. They achieve real-time rendering rates by executing the lighting model on the GPU as a shader program.

#### *4.1.7 Heuristic Lighting Models*

Neyret [29] avoids costly physical simulations when a-priori knowledge can be used to simulate well-known lighting effects. He presents a surface-based shading model for Cumulus clouds that simulates inter-reflection in concave regions (light traps) and the corona at the silhouette.

Wang [42] does not employ a lighting model at all, but lets an artist associate the cloud's color to its height.

#### *4.1.8 Acceleration Techniques*

**Pre-Computed Radiance Transfer:** The lighting distribution for static volumes under changing lighting conditions can be accelerated by pre-computing the radiance transfer through the model.

Lensch et al. [26] pre-compute the impulse response to incoming light for a mesh-based model. Local lighting effects are simulated by a filter function applied to a light texture. Looking up the pre-computed vertex-to-vertex throughput yields global lighting effects.

Sloan et al. [38] use SH as emitters, simulate their distribution inside the volume and store them as transfer vectors for each voxel. The volume can be rendered efficiently in combination with a shader-based local lighting model under changing light conditions.

**Radiance Caching:** Jarosz et al. [19] use radiance caching to speed up ray marching rendering process. Radiance values and radiance gradients are stored as SH coefficients within the volume and re-used.

## 4.2 Global Illumination

Inter-cloud shadowing and indirect illumination cannot be efficiently simulated by participating media lighting models. These effects require global illumination techniques, usually applied in surface-based scenes.

#### *4.2.1 Shadows*

Sphere-based scene representations allow a fast shadow calculation, either, e.g., by using SH for representing the distribution of blocking geometries [34] (Fig. 8, right), or by accumulating shadows in image space [37].

The CUDA implementation of the GigaVoxel algorithm [8] allows to render semi-transparent voxels and to cast shadow rays. Employing the inherent cone tracing capability produces soft shadows.

#### *4.2.2 Indirect Illumination*

Voxel cone tracing applied to a sparse octree on the GPU is used by Crassin et al. [9] for estimating indirect illumination. The illumination at a surface point is evaluated by sampling the neighborhood along a small number of directions along cones.

A fast ray-voxel intersection test for an octree-based scene representation is used in Thiedemann et al. [40] for estimating near-field global illumination.

## 5 SUMMARY AND OUTLOOK

In the following table we summarize efficient lighting and rendering techniques for clouds with different types of volume (rows) and boundary appearances (columns).

| | diffuse | fractal | sharp |
|---|---|---|---|
| dense | [6, 7, 10, 15] [16, 35, 44] | [4, 7, 11] [17, 22] | [4, 42] |
| thin | [22, 36, 44] | [11, 17, 36] | do not exist |
| layer | [3] | | |

The approaches surveyed represent a set of impressive but specialized solutions that employ fairly diverse techniques. However, all known approaches cover only some of the phenomena found in nature (see Sec. 1.1). Extensive memory usage or the algorithm's complexity often limit the size of a cloud volume. Thus, the realistic visualization of real-world cloud scenes still is and will remain widely open for future research for years to come.

**Major future challenges** to be tackled include the efficient handling of

- heterogeneous cloud scene rendering, i.e., the simultaneous visualization of different cloud types, each favoring a specific cloud representation, lighting and rendering technique;
- large-scale cloud scenes, e.g., clouds over Europe;
- cloud-to-cloud shadows and cloud inter-reflection, i.e., the combination of global illumination techniques with participating media rendering;
- the inclusion of atmospheric scattering models for lighting clouds;
- cloud rendering at different scales, i.e., views of clouds from close and far, and seamless transitions in between, requiring continuous LOD techniques;
- temporal cloud animations implying dynamic volumes and employing cloud simulation models.

## ACKNOWLEDGEMENTS

Work supported by Austrian FFG Grant #830029.

## 6 REFERENCES

[1] D. Bernabei, F. Ganovelli, N. Pietroni, P. Cignoni, S. Pattanaik, and R. Scopigno. Real-time single scattering inside inhomogeneous materials. *The Visual Computer*, 26(6-8):583–593, 2010.

[2] A. Bouthors and F. Neyret. Modeling clouds shape. In *Eurographics, Short Presentations*, 2004.

[3] A. Bouthors, F. Neyret, and S. Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In *EG Workshop on Natural Phenomena*, Vienna, Austria, 2006. Eurographics.

[4] A. Bouthors, F. Neyret, N. Max, É. Bruneton, and C. Crassin. Interactive multiple anisotropic scattering in clouds. In *Proc. ACM Symp. on Interactive 3D Graph. and Games*, 2008.

[5] E. Cerezo, F. Perez-Cazorla, X. Pueyo, F. Seron, and F. X. Sillion. A survey on participating media rendering techniques. *The Visual Computer*, 2005.

[6] D. Cha, S. Son, and I. Ihm. GPU-assisted high quality particle rendering. *Comp. Graph. Forum*, 28(4):1247–1255, 2009.

[7] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. ACM SIGGRAPH Symp. on Interactive 3D Graph. and Games (I3D)*, Boston, MA, Etats-Unis, February 2009. ACM, ACM Press.

[8] C. Crassin, F. Neyret, M. Sainz, and E. Eisemann. *GPU Pro*, chapter X.3: Efficient Rendering of Highly Detailed Volumetric Scenes with GigaVoxels, pages 643–676. AK Peters, 2010.

[9] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. Interactive indirect illumination using voxel cone tracing. *Comp. Graph. Forum (Proc. Pacific Graph. 2011)*, 30(7), 2011.

[10] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *Comp. Graphics (SIGGRAPH '00 Proc.)*, pages 19–28, 2000.

[11] D. S. Ebert, F. Musgrave, P. Peachey, K. Perlin, and S. Worley. *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 3rd edition, 2003.

[12] V. Forest, L. Barthe, and M. Paulin. Real-time hierarchical binary-scene voxelization. *Journal of Graphics, GPU, & Game Tools*, 14(3):21–34, 2009.

[13] E. Gobbetti, F. Marton, and J. A. Iglesias Guitián. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7-9):797–806, 2008. Proc. CGI 2008.

[14] O. Gourmel, A. Pajot, M. Paulin, L. Barthe, and P. Poulin. Fitted BVH for fast raytracing of metaballs. *Comp. Graph. Forum*, 29(2):281–288, May 2010.

[15] M. J. Harris, W. V. Baxter, Th. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In *Proc. SIGGRAPH/Eurographics '03 Conference on Graph. Hardware*, pages 92–101. Eurographics Association, 2003.

[16] M. J. Harris and A. Lastra. Real-time cloud rendering. *Comput. Graph. Forum (Proc. IEEE Eurographics '01)*, 20(3):76–84, 2001.

[17] K. Hegeman, M. Ashikhmin, and S. Premože. A lighting model for general participating media. In *Proc. Symp. on Interactive 3D Graph. and Games (I3D '05)*, pages 117–124, New York, NY, USA, 2005. ACM.

[18] R. Hufnagel, M. Held, and F. Schröder. Large-scale, realistic cloud visualization based on weather forecast data. In *Proc. IASTED Int. Conf. Comput. Graph. and Imaging (CGIM'07)*, pages 54–59, February 2007.

[19] W. Jarosz, C. Donner, M. Zwicker, and H. W. Jensen. Radiance caching for participating media. *ACM Trans. Graph.*, 27(1):1–11, 2008.

[20] W. Jarosz, D. Nowrouzezahrai, R. Thomas, P. P. Sloan, and M. Zwicker. Progressive photon beams. *ACM Trans. Graph. (SIGGRAPH Asia 2011)*, 30(6):181:1–181:12, December 2011.

[21] C. Knaus and M. Zwicker. Progressive photon mapping: A probabilistic approach. *ACM Trans. Graph.*, 30(3):25:1–25:13, May 2011.

[22] J. M. Kniss, S. Premože, Ch. D. Hansen, and D. S. Ebert. Interactive translucent volume rendering and procedural modeling. In *Proc. IEEE Visualization '02*, pages 109–116. IEEE Comput. Society Press, 2002.

[23] J. M. Kniss, S. Premože, Ch. D. Hansen, P. Shirley, and A. McPherson. A model for volume lighting and modeling. *IEEE Trans. Visualiz. Comput. Graph.*, 9(2):150–162, 2003.

[24] H.-J. Koppert, F. Schröder, E. Hergenröther, M. Lux, and A. Trembilski. 3d visualization in daily operation at the dwd. In *Proc. ECMWF Worksh. on Meteorolog. Operat. Syst.*, 1998.

[25] S. Laine and T. Karras. Efficient sparse voxel octrees.