Real-Time Rendering of Cloudy Natural Phenomena with Hierarchical Depth Impostors

Tamás Umenhoffer, László Szirmay-Kalos

Department of Control Engineering and Information Technology, Budapest University of Technology, Hungary
Email: szirmay@iit.bme.hu

Abstract

This paper presents a real-time method to realistically render dynamic participating media under changing lighting conditions. In order to cope with performance requirements, the volume is built of instances of particle blocks. The simulation and rendering happen on two levels, on the block level and on the volume level. On the volume level blocks are replaced by depth impostors, which allows for very fast recalculation of the cloud illumination. Including depth information into block impostors our technique also eliminates billboard clipping artifacts when the participating medium contains objects. The proposed method can render swirling clouds and smoke on high frame rates, and can be used in real-time applications.

1. Introduction

Participating media [Bli82] are often represented as particle systems [Ree83]. Particle system rendering methods usually execute a pass for each light source to calculate shadows and lighting in a view independent way, and a final gathering pass to compute the image from the camera [Har02]. In these passes particles are splat onto the screen, which substitutes them with a semi-transparent, camera-aligned rectangle, called *billboard* [Sch95]. Ignoring the extension along the third dimension simplifies rendering, but cause visual artifacts when the billboard rectangle intersects an object.

A particle system is a discretization of a continuous volume, which allows us to replace the differentials of the volumetric rendering equation by finite differences. Denoting the length of the ray segment intersecting the sphere of particle j by Δs_j, and the *density*, *albedo* and *phase function* of this particle by τ_j, a_j, P_j, respectively, we obtain the following equation expressing *outgoing radiance* $L(j,\vec{\omega})$ of particle j at direction $\vec{\omega}$:

$$L(j,\vec{\omega}) = I(j,\vec{\omega}) \cdot (1-\alpha_j) + \alpha_j \cdot C_j, \qquad (1)$$

where $I(j,\vec{\omega})$ is the *incoming radiance*, $\alpha_j = 1 - e^{-\tau_j \Delta s_j}$ is the *opacity* that expresses the decrease of radiance caused by this particle due to *extinction*, and

$$C_j = a_j \cdot \int_{\Omega'} I(j,\vec{\omega}') \cdot P_j(\vec{\omega}',\vec{\omega}) d\omega'$$

is the contribution from *in-scattering*. The opacity also depends on the distance of the ray and the particle center, since rays that are close to the center travel longer inside the particle sphere and thus the attenuation is more significant. This dependence is usually represented by a *opacity billboard*, which is also used to display the particle as semi-transparent rectangles perpendicular to the ray.

The in-scattering term requires the evaluation of a directional integral at each particle j. Real-time methods usually simplify this integral and consider only the directions of the light sources in these integrals, and thus allowing only attenuation and forward scattering [Har02]. The incoming radiance for all particles can be effectively evaluated executing a light pass for each light source. In a light pass, particles are sorted along the light direction, and their opacity billboards are rendered one by one in this order. Before rendering a particle, the color buffer is read back to obtain the light attenuation at this particle, then the opacity texture of the particle is combined with the image to prepare the light attenuation for the subsequent particle. The incoming radiance of a particle due to a given light source is computed from the light source intensity and the opacity accumulated so far.

If we know the incoming radiance of particles, the volume can efficiently be rendered from the camera using alpha blending. The in-scattering term of a particle is obtained by multiplying the albedo and the phase function with the incoming radiance values due to the different light sources, which is attenuated according to the total opacity of the particles that are between the camera and this particle. This requires the sorting of particles in the view direction before sending them to the frame buffer in back to front order. At a given particle, the evolving image is decreased according to the opacity of the particle and increased by its in-scattering term (equation 1).

In this paper we propose the application of particle hierarchies to reduce the computational burden of rendering. On the lower level, particles are grouped into blocks, that are rendered once for the current viewing or lighting direction, then the role of the individual particles are taken by these blocks. To eliminate the read-backs of the color buffer, we compute the attenuation at discrete depth samples during light passes. We also propose a novel solution for including objects into the volume without billboard clipping artifacts, which uses *depth impostors*, i.e. billboards augmented with depth information. Unlike *nailboards* [Sch97, Szi05], our depth impostors store both the front and back depths of a block.

2. The new method using particle hierarchies

To reduce the computational burden of rendering, particle hierarchies are formed, and the full system is built of smaller similar blocks. As most natural phenomena shows self-similarity, we can use this approximation in most cases.

A single *block* represents particles that are close to each other. Before rendering for a given direction, the image of the particles of a block is determined from this direction, and then we use these images instead of individual particles. The image is called *depth impostor*. A pixel of the depth impostor stores information that is needed about the particles which project onto this pixel, particularly their total opacity, their minimum (front) and maximum (back) depths. During rendering an impostor pixel acts as a "super-particle" that concentrates all those particles of the block, which are projected onto it. The total opacity is used in the radiance transfer, while the depths are taken into account to eliminate artifacts caused by objects included in the volume.

Replacing particles by blocks, the computation burden can be reduced significantly. If we want a system with N particles, we can build a block of b particles and instance it N/b times. The hierarchical approach needs only $b+N/b$ calculations during simulation and color computation, in contrast to N calculations of the non-hierarchical method.

2.1. Generating a depth impostor

To generate a depth impostor representing a block for a particular direction (either light or camera), the particles of the block are rendered and the total opacity, and front and back depths are computed for each pixel on the GPU. These depth impostors are first generated for particle spheres and then for volume blocks. The opacity texture of a particle is predefined, and the depth textures of a particle sphere can be created analytically evaluating the depths of a sphere in the preprocessing phase.

The total opacity of a block could be determined using alpha blending of the opacity textures of the particles. The depth values of the block, however, require a different operation. A simple approach would generate the front and back depth textures of a block by rendering particle the front and back textures of the particles, overwriting the fragment depth value in the fragment shader, and letting the z-buffer to find the minimum and the maximum in two different passes. However, this simple approach needs three rendering passes for each particle block. Fortunately, it is also possible to execute the three different calculations in a single rendering pass if depth testing is replaced by alpha testing. With the GL_EXT_blend_minmax extension we can set a blend function which computes minimum or maximum. However, this blending is appropriate only for the depth values, but not for the total opacity, which can be solved by the GL_EXT_blend_equation_separate extension, allowing a different blending type for the alpha channel. The layers of a depth impostor are shown in figure 1.

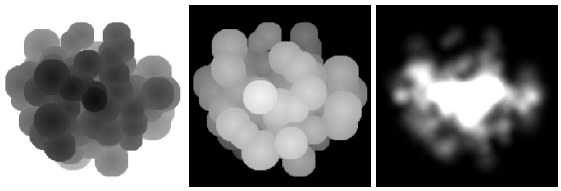

Figure 1: *Depth impostor layers of a block: front depth, back depth, and accumulated opacity*

2.2. Using depth impostors during light passes

Rendering participating media consists of a separate light pass for each light source determining the approximation of the in-scattering term caused by this particular light source, and a final gathering step. Handling light volume interaction on the particle level would be too computation intensive since the light pass requires the incremental rendering of all particles and the read back of the actual result for each of them. To speed up the process, we render particle blocks one by one, and separate light–volume interaction calculation from the particles.

During a light pass we classify particle blocks into groups

according to their distances from the light source, and store the evolving image in textures at given sample distances. These textures are called *slices*. The first texture will display the accumulated opacity of the first group of particle blocks, the second will show the opacity of the first and second groups of particle blocks and so on. The required number of depth samples depends on the particle count and the cloud shape. For a roughly spherical shape and relatively few particles (where overlapping is not dominant), even four depths can be enough. Figure 2 shows this technique with five depth slices. Four slices can be computed simultaneously if we store the slices in the color channels of one RGBA texture. For a given particle, the vertex shader will decide in which slice (or color channel) this particle should be rendered. The vertex shader sets the color channels corresponding to other slices to zero, and the pixel is updated with alpha blending.

2.3. Using depth impostors during final gathering

During final rendering we obtain the depth impostors of the blocks for the viewing direction, sort them and render them one after the other in back to front order. The in-scattering term is obtained from the sampled textures of the slices that enclose the pixel of the block (figure 2).

The accumulated opacity of the slices can be used to determine the radiance at the pixels of these slices and finally the reflected radiance of a particle between the slices. Knowing the position of the particles we can decide which two slices enclose it. By linear interpolation between the values read from these two textures we can approximate the attenuation of the light source color. Harris used a similar technique in [HBSL03], where he stored these slices in a 3D texture called oriented light volume. In order to obtain a better multiple scattering approximation, the radiance of those pixels of both enclosing slices are taken into account, for which the phase function is not negligible.

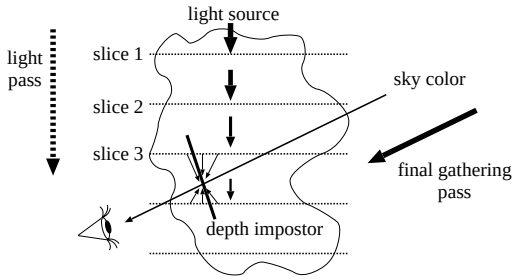

Figure 2: *Final gathering for a block*

2.4. Objects in clouds

The main problem with billboard type particle systems is that billboards are planes, thus they have no extension along one dimension. This can cause artifacts when billboards intersect objects, i.e. the intersection of the billboard plane and the object becomes clearly noticeable (figure 3). The core of this problem is that a billboard fades those objects that are behind it according to its transparency as if the object were fully behind the sphere of the particle. However, those objects that are in front of the billboard plane are not faded at all, thus transparency changes abruptly at the object billboard intersection. This problem is solved using the extension of the particle block, i.e. the interval of the block in the depth direction, which is stored in impostor texels.

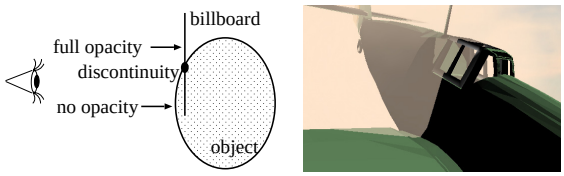

Figure 3: *Problems caused by objects in a volume rendered as billboards. Where the billboard plane intersects the object, transparency becomes discontinuous.*

In order to attack this problem, first we render all objects of the scene and save the depth buffer in a texture. Then the particle blocks are rendered one by one, enabling depth test but disabling depth write. When rendering a particle block, we compute the interval the ray travels inside the block adding the depth value of the block center to the front and back depth values of the depth impostor. This interval is compared with the value storing the depth of the visible object. If the object depth value is outside the interval, then the object is either fully visible or fully occluded by the particle sphere, thus we can rely on the z-buffer and alpha-blending hardware to compute the correct result. However, when the interval of the block encloses the depth of the object, only a part of the volume block occludes the object. In this case, the opacity of the particle block is scaled according to the relative distance between the front depth of the particle block and the object, and the depth interval of the particle block. This scaling corresponds to the assumption that the density is uniform inside a block. The results are shown in figure 4.

Figure 4: *Volume rendered with depth impostors eliminating billboard clipping artifacts*

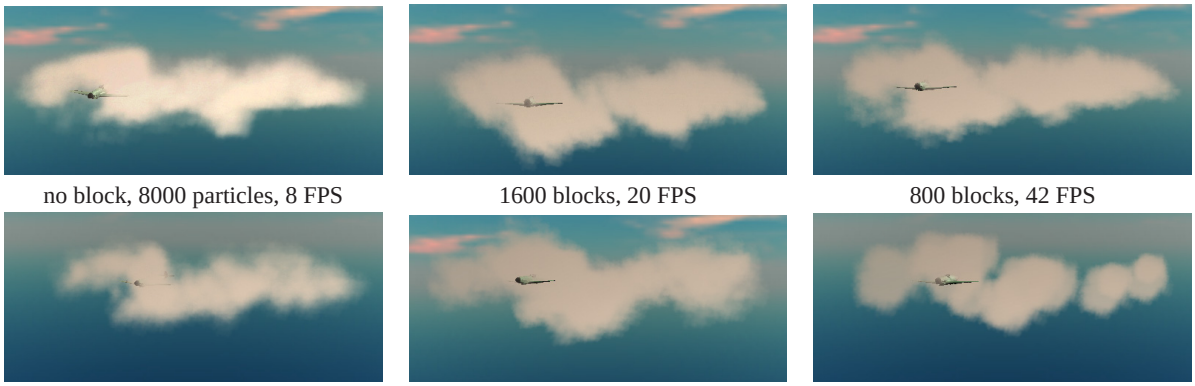

Figure 5: *Images of animated volumes of 8000 particles organized in different number of blocks*

3. Results

The presented algorithm has been implemented in OpenGL/Cg environment on an NV6800GT graphics card. The animated cloud of figure 5 consists of 8000 particles grouped to different number of blocks, and is illuminated by a directional light. The rendering speed increases as we put more particles in a single block until 40 blocks. For higher number of particles per block, rendering gets slower, because of the block computation overhead.

Figure 6: *Swirling cloud rendered at 180 FPS*

4. Conclusions

This paper proposed to build particle clouds of blocks. A block itself represents many particles, and is defined by a depth impostor. We also applied depth sampling to compute self-shadowing and multiple forward scattering quickly. As a combined effect of these improvements, the presented method renders realistically shaded dynamic smoke or cloud formations under changing lighting conditions at high frame rates, taking advantage of the GPU. On the other hand, the inclusion of front and depth information into the depth impostors eliminated billboard clipping artifacts when the volume contains 3D objects.

5. Acknowledgement

This work has been supported by OTKA (T042735), GameTools FP6 (IST-2-004363) project, by the Spanish-Hungarian Fund (E-26/04).

References

[Bli82] BLINN J. F.: Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82 Proceedings* (1982), pp. 21–29. 1

[Har02] HARRIS M. J.: Real-time cloud rendering for games. In *Game Developers Conference* (2002). 1

[HBSL03] HARRIS M. J., BAXTER W. V., SCHEUERMANN T., LASTRA A.: Simulation of cloud dynamics on graphics hardware. In *Eurographics Graphics Hardware'2003* (2003). 3

[Ree83] REEVES W. T.: Particle systems - techniques for modelling a class of fuzzy objects. In *SIGGRAPH '83 Proceedings* (1983), pp. 359–376. 1

[Sch95] SCHAUFLER G.: Dynamically generated impostors. In *I Workshop - Virtual Worlds - Distributed Graphics* (1995), pp. 129–136. 1

[Sch97] SCHAUFLER G.: Nailboards: A rendering primitive for image caching in dynamic scenes. In *Eurographics Workshop on Rendering* (1997), pp. 151–162. 2

[Szi05] SZIJÁRTÓ G.: 2.5 dimensional impostors for realistic trees and forests. In *Game Programming Gems 5*, Pallister K., (Ed.). Charles River Media, 2005, pp. 527–538. 2

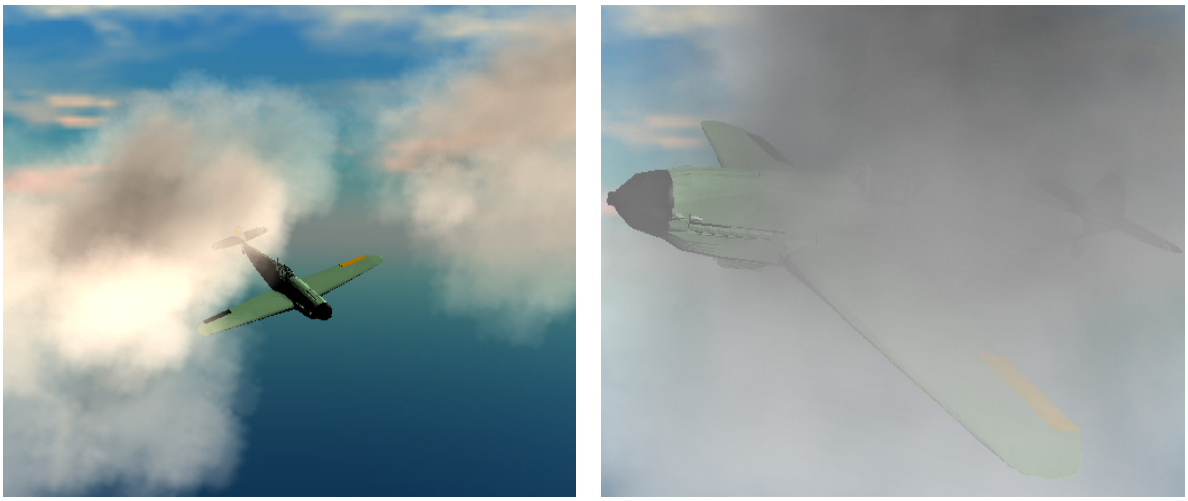

Figure 7: *Swirling cloud crossed by a plane rendered at 180 FPS*