Eurographics Symposium on Rendering 2009
Hendrik P. A. Lensch and Peter-Pike Sloan
(Guest Editors)

Volume 28 (2009), Number 4

GPU-Assisted High Quality Particle Rendering

Deukhyun Cha[†], Sungjin Son[‡], and Insung Ihm[†]
[†]Department of Computer Science and Engineering, Sogang University, Korea
[‡]System Architecture Lab., Samsung Electronics, Korea

Abstract

Visualizing dynamic participating media in particle form by fully solving equations from the light transport theory is a computationally very expensive process. In this paper, we present a computational pipeline for particle volume rendering that is easily accelerated by the current GPU. To fully harness its massively parallel computing power, we transform input particles into a volumetric density field using a GPU-assisted, adaptive density estimation technique that iteratively adapts the smoothing length for local grid cells. Then, the volume data is visualized efficiently based on the volume photon mapping method where our GPU techniques further improve the rendering quality offered by previous implementations while performing rendering computation in acceptable time. It is demonstrated that high quality volume renderings can be easily produced from large particle datasets in time frames of a few seconds to less than a minute.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing, I.3.1 [Computer Graphics]: Hardware Architecture—Graphics processors

1. Introduction

1.1. Background and our contribution

Particles are an excellent means of representing dynamic and fuzzy effects of natural phenomena. They are frequently used by animators to effectively model participating media such as smoke and clouds. The generation of physically accurate rendering images from large particle datasets by considering the complicated light transport phenomena often leads to a substantial expense in computation. Among the several direct particle rendering methods that have been employed in the computer graphics community, one effective approach is transforming particle data into corresponding volume data and then applying an advanced volume rendering algorithm that permits effective massively parallel simulation of the light transport theory on today's many-core processors, generating high quality renderings in practical time.

In this paper, we propose several computing techniques for particle volume rendering that can be easily accelerated by the current graphics processing unit (GPU). Our method is based on the volume photon mapping algorithm [Jen01] that is routinely used to produce high quality volume rendering images. We first present a three-pass density estimation method, exploiting the GPU's fast rectangle drawing capability, to use adaptive smoothing lengths for transforming given large particle datasets into volumetric density fields that are then visualized in the following rendering pipeline. Second, we demonstrate that the mathematically correct numerical method for tracing photons in nonhomogeneous media can be implemented practically on the GPU, faithfully simulating multiple scattering, which is one of the most intractable phenomena to reproduce in volume rendering. Then, we show that the speed of volume rendering can be increased drastically by caching precomputed in-scattered radiances in a simple 3D grid structure, and reusing them for fast radiance resampling. Finally, we show that noise can be naturally and efficiently included in the ray marching stage of our GPU-assisted volume photon mapping framework, which allows animators to easily model fuzzy appearances of participating media at very small expense.

The fundamental goal of this work is to provide animators in the special effects industry with a fast volume rendering tool that can significantly increase the efficiency of their animation process. Unlike some of the recent works that pursue real-time volume rendering by possibly performing substantial preprocessing and/or employing simpler light transport models, the presented rendering scheme takes the other direction, attempting to further improve rendering quality provided by the frequently used volume photon mapping technique. We demonstrate that high quality volume renderings

can be easily created from large particle datasets in time frames of a few seconds to less than a minute.

2. Previous work

Since introduced to the computer graphics community [Ree83], particles have been employed effectively to represent dynamic and fuzzy phenomena of participating media (for instance, refer to an application of particles in the special effects industry for generating volumetric effects [Kap03]). In the physically-based fluid simulation, the Lagrangian particles have also been explored as a means of complementing the Eulerian grid-based approaches [Mon88]. In many particle systems, the rendering computation was usually performed by splatting particles one by one onto screen. While often satisfactory, such direct particle drawing methods were not very appropriate for creating global illumination effects.

Following the pioneering works [Bli82, KH84], there have been extensive studies on the realistic rendering of participating media (please refer to the survey paper by Cerezo for a comprehensive overview of this theme [CPP*05]). Among them, volume photon mapping has been frequently used because of its simple and practical simulation of the complex light transport process through volume photon maps [JC98, Jen01]. In order to ease the computational overhead of the volume photon mapping method, the idea of radiance caching was recently extended to participating media [JDZJ08]. Photon tracing was also performed on the face-centered cubic lattice to simulate the light transport phenomena [QXF*07].

In an aim to achieve interactive or real-time rendering of participating media, various approximation models were implemented on modern graphics hardware [DYN02, KPh*03, REK*04, SRNN05]. Fast rendering was also possible through precomputations of light transport information [HL01, PARN04, HAP05, SKSU05, ZRL*08, BNM*08, CNLE09].

3. GPU-assisted particle rendering

3.1. Rendering pipeline overview

The presented particle rendering scheme consists of four major stages: *density volume generation*, *volume photon tracing*, *illumination cache construction*, and finally *ray marching*. It starts by transforming the input particles into a volumetric density field for efficient rendering computation on the GPU. Particles are an excellent means for modeling participating media. From the perspective of physically-based volume rendering that needs repeated resampling operations over sampled density data, however, grids are better suited for GPU computation than particles, as the GPU has been optimized for grid structures. For dynamic particle datasets, the conversion has to be carried out for each animation frame, demanding a fast transformation process. In order to accelerate it, we used a rectangle drawing-based method that provides both fast and adaptive conversion on the GPU.

Then, a volume photon map was built by tracing photons stochastically within participating media. While most participating media we routinely handle are nonhomogeneous, a simple propagation distance estimation model that only works for homogeneous media has been usually applied for practical volume rendering, often producing erroneous illumination effects. In order to achieve high quality renderings, we solved on the GPU the integral equation derived for nonhomogeneous media, which offers mathematically correct simulation at small extra cost.

When the volume photon map was ready, radiance estimation computation was performed before ray marching started, by building a 3D grid structure, called *illumination cache*. This grid, which stored sampled radiance in-scattered towards the viewing direction, was then exploited by the GPU to quickly reconstruct incoming radiance at sampled points through trilinear interpolation in the ray marching stage. Finally, ray marching computation was performed to generate the final rendering images, efficiently utilizing the density volume and illumination cache. In this ray marching stage, we demonstrated that clever use of precomputed Perlin noise could add fine detail to particle data at insignificant expense, which enabled animators to easily model the fuzzy, natural looking appearance of participating media.

3.2. Adaptive density volume generation

In smoothed particle hydrodynamics (SPH), the physical properties of a given particle system are reconstructed using the kernel function [Mon88]. In particular, the density of particle p is estimated at its location as a weighted sum $\rho(\boldsymbol{p}_p) = \sum_q m_q \cdot W(|\boldsymbol{p}_p - \boldsymbol{p}_q|, h_p)$ over neighboring particles q with mass m_q, where $\boldsymbol{p}_a$ is the location of particle a, and $W(x,h)$ is a smoothing kernel with smoothing length of h and circular support domain with radius κh (note that the value of κ may not be one).

By evaluating the weighted sum at grid points, particle data can be easily transformed into corresponding volumetric data. An important parameter in this process that has significant influence on the final renderings is the smoothing length h_{ijk}, which is applied at each grid point $\boldsymbol{p}_{ijk}$. It is known that local accuracy achieved by density estimation depends on the number of neighboring particles involved. When too few are used, it may result in low accuracy. On the other hand, if too many are used, local properties may be smoothed out despite the high computational effort. In order to gain a consistent level of accuracy throughout the entire simulation domain, the smoothing length should be chosen by reflecting the local particle density, so that a sufficient and necessary number of particles are used.

Our rendering system provides both uniform and adaptive smoothing length methods that are implemented using sim-

ple rectangle drawing that is easily accelerated on the GPU. The single-pass uniform length method that has been frequently employed in various computer graphics applications is computationally cheap, whereas the multi-pass adaptive method offers automatic adaptation for datasets of irregular distribution with steep density contrasts.

Note that the kd-tree structure that provides effective nearest neighbor searching may be employed for the adaptive density volume generation. Recently, a kd-tree construction algorithm was presented for the GPU using the CUDA API [ZHWG08]. In this work, an iterative k-nearest neighbor search algorithm was implemented based on range searching because the often-used priority queue method [Jen01] was not well-suited for the CUDA's computing architecture. While proven to be effective for caustics rendering, their construction algorithm is somewhat complicated to program on the GPU. In stead of implementing the kd-tree method, we decided to develop an approximate searching methods that, whether adaptive or not, demands drawing of only $O(N)$ rectangles for N particles (under the assumption that $h^{(0)}$ is small compared with the entire simulation domain) and allows a much simpler implementation on the GPU than the kd-tree approach.

3.2.1. Density estimation with uniform smoothing length

Suppose that a volume with resolution of $n_x \times n_y \times n_z$ is to be constructed with uniform smoothing length h. In the preparation step, input particles traverse the CPU to create n_z bins, one per $z = z_i$ slice ($0 \le i < n_z$), each holding particles that may affect grid points on the corresponding xy slice. Stored as vertex arrays, the particle bins are sent to the GPU for fast processing, where particles are orthogonally projected bin-by-bin onto the image plane by drawing squares of size $2\kappa h$, centered at their respective positions in the volume space. For each rasterized pixel that corresponds to a grid point, the pixel shader calculates the distance to the current particle, and accumulates the appropriate kernel-weighted mass. After the rectangle drawing process is repeated over all bins, the resulting n_z images are assembled into a volumetric density field in the form of a 3D texture.

3.2.2. Density estimation with adaptive smoothing length

For adaptive selection of smoothing length, we have devised a predictor-corrector scheme that proceeds in three steps. In this method, the smoothing length is controlled by a user-specified parameter N^*, which intends that a rough number of particles be involved at each grid point $\boldsymbol{p}_{ijk}$ in the density estimation. In order to avoid a smoothing length that is too large in low-density region, it is limited by the maximum length $h^{(0)}$, which is also set by the user.

[Step 1: Prediction] In the beginning, one rectangle drawing pass is carried out with an initial length $h^{(0)}$, counting the number of particles, $N_{ijk}^{(0)}$ that exist around grid point $\boldsymbol{p}_{ijk}$ in the spherical volume with radius $\kappa h^{(0)}$. After the particle projection is finished, the average density at $\boldsymbol{p}_{ijk}$ is estimated as $\bar{\rho}_{ijk} = \frac{mN_{ijk}^{(0)}}{\frac{4}{3}\pi(\kappa h^{(0)})^3}$ under the assumption that the mass m is constant. By letting $\bar{\rho}_{ijk} = \frac{mN^*}{\frac{4}{3}\pi(\kappa h_{ijk}^{(1)})^3}$, a new smoothing length $h_{ijk}^{(1)}$ that, hopefully, contains N^* particles in the supporting domain is estimated as $h_{ijk}^{(1)} \equiv \min\{h^{(0)}\left(\frac{N^*}{N_{ijk}^{(0)}}\right)^{\frac{1}{3}}, h^{(0)}\}$.

[Step 2: Correction] The estimate $h_{ijk}^{(1)}$ is reliable only when particle distribution is varying smoothly. In reality, particles are distributed in a complicated and irregular manner, hence a correction computation must be performed to improve accuracy. In this second step, another round of rectangle drawing is performed, counting the number of particles, $N_{ijk}^{(1)}$ found with $h_{ijk}^{(1)}$. When the smoothing length has been underestimated, that is, $N_{ijk}^{(1)} < N^*$, $h_{ijk}^{(1)}$ must be increased appropriately. The correction can be done from a simple relation $\frac{N_{ijk}^{(0)} - N_{ijk}^{(1)}}{\frac{4}{3}\pi(\kappa h^{(0)})^3 - \frac{4}{3}\pi(\kappa h_{ijk}^{(1)})^3} = \frac{N^* - N_{ijk}^{(1)}}{\frac{4}{3}\pi(\kappa h_{ijk}^*)^3 - \frac{4}{3}\pi(\kappa h_{ijk}^{(1)})^3}$, leading to a new estimate

$$h_{ijk}^* \equiv \left(\frac{(N^* - N_{ijk}^{(1)})((h^{(0)})^3 - (h_{ijk}^{(1)})^3)}{N_{ijk}^{(0)} - N_{ijk}^{(1)}} + (h_{ijk}^{(1)})^3\right)^{\frac{1}{3}}.$$

If the smoothing length has been overestimated, that is, $N_{ijk}^{(1)} > N^*$, it is adjusted as $h_{ijk}^* \equiv h_{ijk}^{(1)}\left(\frac{N^*}{N_{ijk}^{(1)}}\right)^{\frac{1}{3}}$, reflecting the local particle distribution. In this case of overestimation, the estimate may be further refined by multiplying a relaxation coefficient $\alpha = (1 - s(\frac{N_{ijk}^{(1)}}{N_{ijk}^{(0)}}))$ as $h_{ijk}^* \equiv \alpha h_{ijk}^{(1)}\left(\frac{N^*}{N_{ijk}^{(1)}}\right)^{\frac{1}{3}}$ to reflect gradient of particle distribution, in which s is a user-defined scale factor in $[0, 1]$.

[Step 3: Density computation] In the final step, the volume density field is generated by performing one more pass of rectangle drawing using the adaptively chosen smoothing distance h_{ijk}^*.

3.3. Test results

Table 1 shows timing statistics that compare the run times of several methods. First, we coded a simple uniform grid method (UG), similar to the GPU implementation designed for photon mapping [PDC*03], where particles are organized into grid cells of size κh and the cell containing given volume grid point $\boldsymbol{p}_{ijk}$ and its 26 neighbors are queried. Then, this uniform smoothing length method was tested over our rectangle drawing techniques, where USL and ASL3

respectively denote the uniform and adaptive methods described in this section. We also programmed a two-pass adaptive method (ASL2) that skips the correction step, and hence is faster, but possibly less accurate.

(unit: sec.)

Resolution	$h^{(0)}$	UG	USL	ASL2	ASL3
129×103×107	0.96	1.38	0.53	0.92	1.20
257×206×213	0.96	5.48	1.84	3.55	4.66
257×206×213	1.44	10.02	4.53	7.47	9.98

Table 1: *Timing statistics measured on an NVIDIA GeForce GTX 280 GPU. A physically-based simulated scene in Figure 1 that contains 326,752 particles was tested to compare run times of various density estimation methods. The uniform grid (UG) and uniform smoothing length (USL) methods used fixed smoothing length. Despite being faster than the other adaptive methods, they were unable to adapt themselves to the density of the particle data. With additional expenses, the presented three-pass method (ALS3) was capable of adaptively controlling smoothing lengths, entailing enhanced renderings, as shown in Figure 1. All the timings contain the CPU time spent for building the uniform grid and slice bins, respectively.*

As the test results indicate, our uniform length method (USL) compares very favorably with the uniform grid method (UG) that usually took longer than the sophisticated, three-pass adaptive method (ASL3). The run time of our multi-pass method greatly depends on the initial choice of $h^{(0)}$ that determines the rectangle size. Recall that our scheme controls the overall accuracy of density estimation using the target particle number, N^*. Initially, $h^{(0)}$ is set large enough, so that each grid point holds at least N^* particles in its supporting domain. A careless, fixed choice of $h^{(0)}$, however, may result in a waste of time as it can be too large for most of the grid points. It is possible to select $h^{(0)}$ heuristically by applying a simple and fast histogram technique on the CPU. The figures in the ASL3 column indicate run times obtained when $h^{(0)}$ was adaptively set to $0.75h^{(0)}$ in the high-density region, in which the statistics in Table 2 are the same as when fixed $h^{(0)}$ was applied.

In addition to the timing performance, we have also investigated the actual number of particles used by our multi-pass methods. In Table 2, N_{avg}^{used} and N_{stdev}^{used} respectively indicate the average and standard deviation of the number of particles found at grid points for density estimation. As expected for the uniform smoothing length method (USL), the average numbers increased quickly as $h^{(0)}$ became larger, demanding more computing time. When the search radius was small, the standard deviation was large, entailing an uncontrolled accuracy in density estimation. The variation was reduced when a larger smoothing distance was used, but too many particles tended to be found in the high-density region, which caused highly detailed features to be blurred. This phenomenon was quite expectable because the tested particle data have a very irregular distribution with steep density contrasts, as displayed in Figure 1.

	$h^{(0)}$	N^*	N_{avg}^{used}	N_{stdev}^{used}	$\frac{N_{stdev}^{used}}{N_{avg}^{used}}$
USL	0.24	-	11.08	15.42	1.39
	0.48	-	74.37	55.50	0.74
	0.96	-	447.49	250.21	0.55
	1.44	-	1282.80	667.91	0.52
ASL2	0.96	32	61.32	30.65	0.49
		64	98.14	41.47	0.42
	1.44	32	80.21	41.25	0.51
		64	136.66	59.76	0.43
ASL3	0.96	32	33.05	10.14	0.30
		64	62.21	12.22	0.19
	1.44	32	34.65	8.51	0.24
		64	67.00	13.03	0.19

Table 2: *Statistics on the number of particles used at each grid point for density estimation. The N_{stdev}^{used} column revealed that the range of particle numbers was too wide when the uniform smoothing length method (USL) was applied, resulting in irregular accuracy in density estimation. On the other hand, the variation reduces markedly thorough automatic smoothing length control by the presented adaptive method (ASL3). In these statistics, we only considered grid points that contained at least one particle in their cells.*

By applying the three-pass adaptive method (ASL3), however, we gained much more control over the number of particles, as shown in the N_{avg}^{used} and N_{stdev}^{used} columns. In particular, the figures in the ASL2 and ASL3 rows indicate that the additional correction step deserved extra computing time. It may seem that the standard deviation is still high. Considering that the tested particle dataset was very irregular in its distribution, however, we believe that a good enough control for effective rendering was achieved. This fact is well observed in the rendering images in Figure 1. When a fixed length method is used, animators usually have to go through a trial and error process of finding the proper fixed smoothing length. On the other hand, our multi-pass methods automatically control the smoothing length, given an initial value of N^*, by appropriately reflecting the particle density (see the caption of Figure 1).

3.4. Volume photon tracing for nonhomogeneous media

3.4.1. Solving integral equation for distance generation

When a volume photon is scattered at location $\boldsymbol{p}_0 \in R^3$ in a given participating medium, a pair of random direction $\vec{\omega}_{next}$ and distance d_{next} to the next interaction point must be stochastically generated. $\vec{\omega}_{next}$ is usually computed by importance-sampling phase function set to the medium. On the other hand, d_{next} has often been generated in volume rendering using uniform random variable $\xi \in (0,1)$ as $d_{next} = -\frac{\log \xi}{\sigma_t}$ (Eq. (1)), where σ_t is the extinction coefficient at $\boldsymbol{p}_0$ [Jen01].

This expression assumes an exponential distribution with the fixed rate parameter σ_t along the half-closed line $\boldsymbol{p}(x) =$

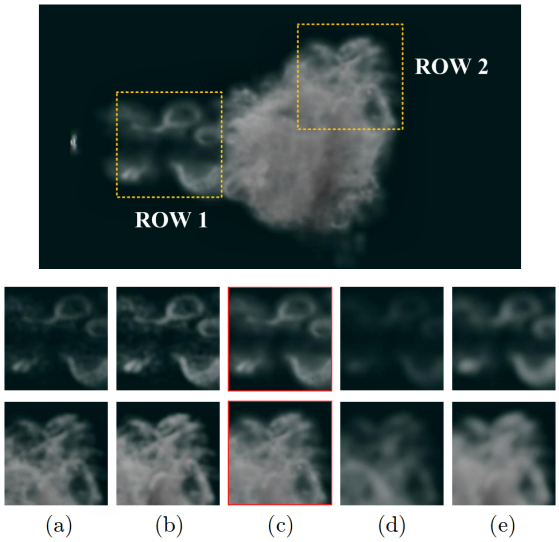


Figure 1: *Comparisons of rendering images produced by uniform and adaptive smoothing lengths. The uniform method (USL) was applied to create images in (a), (b), (d) and (e) using the smoothing length $h^{(0)}$ = 0.48, 0.48, 0.96 and 0.96 and particle mass m = 0.004, 0.008, 0.001 and 0.004, respectively. On the other hand, the presented three-pass technique (ASL3) was applied with $h^{(0)} = 0.96$ and $m = 0.006$ to get the images in (c). As depicted in (c), adaptive selection of the smoothing length smoothes out well the low-density region, while preserving fine details in the high-density region. When the uniform smoothing length method was applied, setting the length too small produced bumpy and noisy renderings ((a) & (b)), whereas setting it too large often entailed excessive blurring, losing details in particle data ((d) & (e)).*

$\boldsymbol{p}_0 + x \cdot \vec{\omega}_{next}$ $(x \geq 0)$, implying that σ_t, the extinction coefficient in volume rendering, must be constant. In general, this assumption does not hold because the extinction coefficient varies very irregularly in nonhomogeneous media, leading to erroneous simulation of light transport phenomena. Based on the principle of the inversion method, the propagation distance d_{next} for nonhomogeneous media with varying extinction coefficient $\sigma_t(\boldsymbol{p}(x))$ can be generated by solving an integral equation as follows: *For a uniform random number* ξ *between 0 and 1, find* d_{next} *such that* $\xi = 1 - e^{-\int_0^{d_{next}} \sigma_t(s)ds}$, *i.e.,* $\int_0^{d_{next}} \sigma_t(s)\,ds = -\ln(1-\xi)$ (Eq. (2)).

This integral equation can be solved numerically by incrementally evaluating the integral along the line $\boldsymbol{p}(x)$ until the sum exceeds $-\ln(1-\xi)$ (this corresponds to marching along the parameterized line). If the accumulated value has not reached $-\ln(1-\xi)$ when the parameter variable x arrives at the boundary of volume, it means that the traveling photon leaves the media without an interaction.

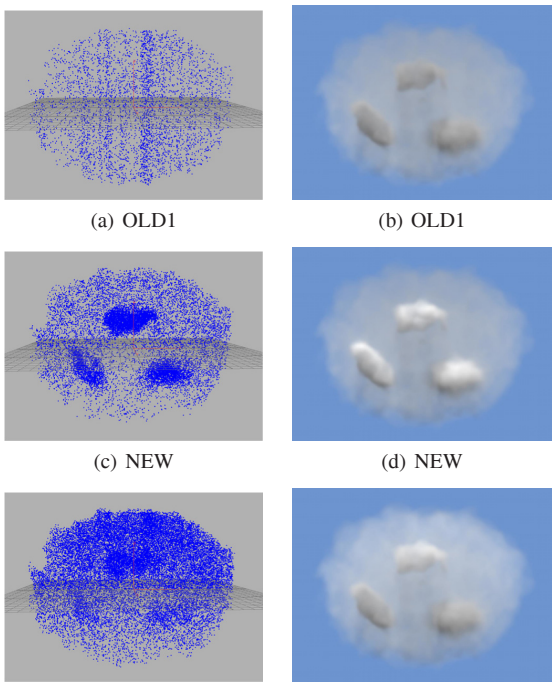
Photon propagation by Eq. (2) has already been reported in previous works, for instance, [JC98, PKK00]. To the best of our knowledge, however, the simple exponential distribution is still in popular use for practical volume rendering, probably due to its computational simplicity (for example, path distance was sampled according to the exponential distribution in a recent Monte Carlo simulation of multiple scattering in nonhomogeneous participating media [JDZJ08]). However, the massively parallel computing power of the recent GPU enabled us to employ the sophisticated numerical model that permits more precise light simulation in volume rendering, as will be analyzed shortly.

3.4.2. Test results

As demonstrated in Figure 2, the stochastically correct scheme compares very favorably with the previous simple one. In this example, a cloud ball of low density, containing three smaller chunks of higher densities, was lit by a light source from above. When photons were traced with Eq. (1) in the cloud whose extinction coefficient was set to its density (OLD1), most propagation distances were overestimated because of the small value of σ_t in the low-density region, forcing the photons to leave the cloud without interacting with the thicker regions. This phenomenon was well reflected in the experimental statistics, where more than four million photons had to be emitted to store only 117,757 volume photons (see the OLD1 column in the table in Figure 2(g)). On the other hand, the mathematically correct method (NEW) nicely reflects the density variation of the cloud ball, as demonstrated in the images in (c) and (d). Intuitively correct, photons were stored, stochastically well distributed according to the density distribution, catching the illumination effect properly.

We have also varied σ_t by scaling the extinction coefficient by a factor of 50, and applied Eq. (1) in the photon tracing stage (OLD50). In this setting, too many photons were stored in the upper region of the cloud where light enters, since the small distance values due to the exaggerated extinction coefficients cause most photons to be absorbed without reaching the lower region. Even with the large number of photons stored (416,219 in the OLD50 column of the table in (g)), the two thick balls at the bottom were not illuminated properly, whereas only 191,636 photons as given in the NEW column of the table, were able to correctly create the light scattering effect by solving the integral equation (compare the images in (f) and (d)).

It may seem that the numerical method that needs an inversion of the integral function is much slower than the method based on Eq. (1), as an integral equation must be numerically solved every time a photon interacts with the participating medium. However, as indicated by the run times given in the table in (g), our test on the GPU revealed that the mathematically correct technique is very competitive, especially for intractable situations, because fewer photons were sufficient to faithfully simulate the light transport phenomena. For precise evaluation of the integral, we applied the fourth-order composite Simpson's rule $\int_{x_{2i}}^{x_{2i+2}} \sigma_t(s)\,ds \approx$



(a) OLD1 (b) OLD1

(c) NEW (d) NEW

(e) OLD50 (f) OLD50

	OLD1	OLD50	NEW
$\sigma_t(\boldsymbol{p}(x))$	$d(\boldsymbol{p}(x))\cdot 1$	$d(\boldsymbol{p}(x))\cdot 50$	$d(\boldsymbol{p}(x))\cdot 1$
Emitted	4,403,200	409,600	614,400
Stored	117,757	416,219	191,636
Time (sec.)	3.934	1.359	1.038

(g) Photon tracing statistics

Figure 2: *Comparison of stochastic distance generation methods. The complicated technique (NEW), based on Eq. (2), effectively depicted the illumination effect in a tricky situation as set up in this example, whereas the simpler technique using Eq. (1) had trouble in photon tracing. The timings, measured on an NVIDIA GeForce GTX 280 GPU show that the numerically more expensive method actually turned out to be faster, as it was able to effectively simulate the light transport phenomena with a smaller number of photons. In our implementation, photons were traced on the GPU by drawing rectangles repeatedly and the fourth-order composite Simpson's rule was applied with step size of one eightieth of the diagonal distance of a density grid cell.*

$\frac{\Delta x}{3}\{\sigma_t(x_{2i}) + 4\sigma_t(x_{2i+1}) + \sigma_t(x_{2i+2})\}$, which runs as fast as less precise, low-order numerical methods on the GPU.

3.5. Single and multiple scattering on the GPU

3.5.1. Illumination cache

In volume photon mapping, most of the rendering time is spent in the final, ray marching stage, in which in-scattered radiance must be repeatedly estimated at each ray sample point. Usually, the single scattering part of incoming radiance is separately evaluated by performing another ray marching toward light sources. On the other hand, the radiance due to multiple scattering is approximated by summing weighted fluxes of neighboring photons.

In an attempt to enhance computational efficiency, a radiance caching algorithm [JDZJ08], was recently presented for volume rendering, where global illumination within participating media was simulated by sparsely sampling the single and multiple scattering radiances and caching them for subsequent extrapolation. While novel and effective, the algorithm is not best suited for GPU implementation because managing the caching structure, for instance, is still too complex for the computational model offered by the current GPU.

For simple and efficient computation on the GPU, we used a 3D grid, called illumination cache, to store view-dependent, sampled incoming radiance. Once built into a texture image, in-scattered radiance at ray sample points can be efficiently reconstructed through trilinear interpolation, easily accelerated by texture units of the GPU. In building the illumination cache, we separately accumulate the two radiances from single and multiple scattering. In particular, when the radiance due to multiple scattering was estimated, we applied the rectangle drawing technique, proposed for the density estimation computation, for adaptive volume radiance estimation, rather than implementing an extra search structure like the kd-tree for finding neighboring photons.

3.5.2. Test results

Table 3 lists statistics regarding illumination cache, gathered on a PC with a 3.16 GHz Intel Core 2 Duo CPU and an NVIDIA GeForce GTX 280 GPU. To demonstrate speedups achieved by the presented GPU technique, we have implemented a CPU version of the same rendering algorithm that also employs the illumination cache, built using the kd-tree method. During ray marching for both single scattering accumulation and final ray marching, we used a sampling distance that is one third of the diagonal distance of a grid cell.

As the table in (a) indicates, the single scattering computation, based on the fourth-order composite Simpson's rule, is remarkably efficient on the GPU, achieving computation times that were up to hundreds times faster than on the CPU, because the simple ray marching operation from each grid point of illumination cache was well suited to the GPU's massively parallel streaming computation architecture. The GPU computation also showed substantial speedups over the CPU when multiple scattered radiance was accumulated onto the illumination cache, as shown in table (b). Here, the performance gap between two processors became larger for a bigger search radius, because the rectangle drawing method on the GPU outperformed the software-based kd-tree operation. Finally, the table in (c) shows how effectively the presented illumination cache worked on the GPU, where the figures in parentheses denote the total rendering time. We have also coded another CPU version that did not utilize

(a) Accumulation of single-scattered radiance

	Resolution	CPU	GPU
Cloud1	189×23×200	4.5	0.012
	283×35×300	16.4	0.032
Cloud2	200×107×200	38.6	0.119
	299×160×300	165.5	0.502
Smoke	200×161×167	55.3	0.136
	300×242×250	257.3	0.630

(b) Accumulation of multiple-scattered radiance

	Resolution	CPU	GPU
Cloud1	189×23×200 (2.8)	19.2 (165,087)	0.98 (195,060)
	283×35×300 (2.8)	62.2	1.84
	283×35×300 (5.6)	422.6	9.90
Cloud2	200×107×200 (2.6)	121.3 (190.067)	3.01 (231,505)
	299×160×300 (2.6)	394.1	9.25
	283×35×300 (5.2)	2988.5	53.09
Smoke	200×161×167 (0.8)	17.5 (130.560)	0.94 (141,828)
	300×242×250 (0.8)	56.1	2.08
	300×242×250 (1.6)	417.0	6.85

(c) Ray marching

	Resolution	CPU	GPU
Cloud1	1,024×576 (1)	296.1 (380.1)	0.14 (4.66)
	2,048×1,152 (1)	498.3 (586.9)	0.83 (5.89)
	2,048×1,152 (4)	1,962.6 (2,043.7)	2.72 (8.01)
Cloud2	1,024×576 (1)	356.6 (863.3)	0.21 (12.01)
	2,048×1,152 (1)	1,409.2 (1,985.4)	1.36 (14.06)
	2,048×1,152 (4)	7,605.6 (8,001.8)	4.47 (16.84)
Smoke	1,024×768 (1)	290.0 (506.9)	0.19 (5.59)
	2,048×1,536 (1)	1,118.0 (1,455.9)	1.77 (7.56)
	2,048×1,536 (4)	6,410.5 (6,663.6)	4.38 (10.19)

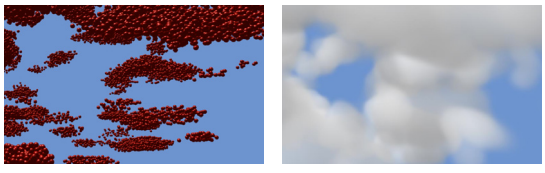
Table 3: *Statistics for the illumination cache. CPU and GPU codes implementing basically the same illumination cache-based rendering algorithm were compared. In (b), the figures in parentheses in the Resolution column denote the maximum search radius used during volume photon gathering, while the other parenthesized numbers represent the respective numbers of photons stored in the volume photon map. In (c), the figures in parentheses in the Resolution column indicate numbers of jittered supersamples per pixel, and the other parenthesized numbers represent total rendering times including the density volume generation. All timings were measured in seconds.*

an illumination cache, but its timings were not included as it ran too slow.

3.6. Ray marching with Perlin noise

Procedural noise has been a very useful tool in volume rendering for increasing the appearance of realism. For instance, volumetric density fields for very natural-looking clouds were generated by adding details through noises to roughly shaped cloud models represented by a set of implicit functions [EMPP03]. The same idea could be easily applied to our GPU-assisted volume rendering scheme in the density estimation stage of our GPU-assisted particle rendering scheme, where pregenerated noise was stored as a texture image for fast access. Another computation point where noise can effectively add extra small-scale detail to already perturbed density fields is in the final ray marching stage.

It has been noted that aliasing artifacts can be reduced by jittering sample points [Jen01]. In fact, we observe that slightly dispersing sample points with noise is a very good way of producing fuzzy effects, particularly for simple modeling data that animators routinely generate, because such particle datasets usually do not contain much detail of participating media, unlike those that are generated by physically-based simulations (see Figure 3). For efficient jittering of sample points on the GPU, we generated offset noises at grid points using Perlin noise in the preprocessing stage, and stored the offset vectors with density value in a four-component 3D texture. Since offset noise to a given sample point is quickly fetched from the texture through trilinear interpolation, the cost of exploiting this position noise in the ray marching step is very cheap.



(a) Particle data (b) Rendering without noise

(c) Density noise added (d) Position noise added

Figure 3: *Perlin noise in ray marching (Cloud2). By slightly jittering sample points during ray marching ((d)), finer detail can be easily added to density in a field that has already been perturbed with noise ((c)). Since the respective scalar and vector noises applied to the density estimation and ray marching stages are stored in 3D textures, the extra computational cost of exploiting noise on our GPU-assisted particle renderer is very cheap.*

4. Experimental results and concluding remarks

We have implemented our GPU-assisted particle rendering method on a PC with a 3.16 GHz Intel Core 2 Duo

CPU and an NVIDIA GeForce GTX 280 GPU, in which the OpenGL and Cg APIs were used for shader programming. In order to show its effectiveness, we have also implemented an equivalent, single threaded CPU version that programmed the same rendering pipeline. Table 4 summarizes the statistics collected for three test scenes (Cloud1: Figure 4, Cloud2: Figure 3, Smoke: Figure 5). The performance results imply that the presented GPU rendering scheme is promising in that it produced high quality renderings in quite affordable time while handling all of such advanced rendering features as adaptively varying smoothing length, stochastically correct photon tracing for nonhomogeneous media, and noise control. The performance gaps between the CPU and GPU implementations became larger as the volume and image resolutions increased.

Figure 4: *Rendering of Cloud1.*

In both implementations, employing the illumination cache is critical because the running times were prohibitive without it. When the same grid resolution was used for storing the sampled density and radiance, the caching technique produced basically no difference in renderings. It should be mentioned, however, that there was a slight difference between the images produced by the two processors mainly due to the different ways of generating random numbers needed for the stochastic photon tracing and noise application (see Figure 5). However, it was hard to tell if one is better than the other except that one was much faster than the other as shown in Table 4.

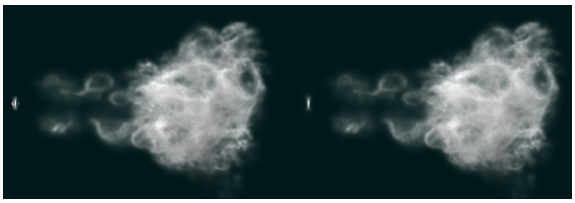


Figure 5: *Rendering comparison (Smoke). Two rendering results by the GPU (left) and the CPU (right) are compared.*

The table in Figure 6 also gives a general idea on how the timing performance of our method is affected by the employed grid resolution. As observed in the table, the volume photon tracing (VPT) and ray marching (RM) stages were rather insensitive to the grid resolution since the step sizes used in the respective numerical integration were the dominant factors in both computations. This is in contrast with the other two stages (DVG and ICC) in which the density and radiance had to be computed respectively for each grid point. While becoming faster, our renderer entailed undesired smoothing of details at lower grid resolutions. Furthermore, when it was too low with respect to the scene domain, jaggies became visible due to insufficient number of grid cells. Our test shows that noise in the ray marching stage is quite useful for reducing such aliasing problem. For the Cloud2 scene, such jaggies were invisible when the resolution was $200 \times 107 \times 201$ or higher (note that this resolution is not that high considering the entire simulation domain).

The presented GPU-assisted particle rendering pipeline

	Cloud1	Cloud2	Smoke
Image size	1024×576	1024×576	1024×768
	2048×1152	2048×1152	2048×1536
Particles	500,000	50,000	326,752
Grid size	$339 \times 42 \times 361$	$299 \times 160 \times 301$	$280 \times 255 \times 233$

		DVG	VPT	ICC	RM	
Cloud1	CPU	15.37	1.40	65.81	43.26	164.58
($h^{(0)} = 1.07$)	GPU	1.90	0.79	2.22	0.55	2.16
Cloud2	CPU	13.97	2.05	307.69	168.21	585.52
($h^{(0)} = 1.6$)	GPU	0.55	0.74	8.29	0.66	2.54
Smoke	CPU	64.17	16.59	175.76	191.77	517.39
($h^{(0)} = 1.44$)	GPU	10.56	1.39	3.90	0.74	2.94

Table 4: *Overall timing performance. The step-by-step dissection of computation times taken by the presented rendering pipeline is shown. In this experiment, the same volume resolution (Grid size) was applied to the creation of density volume (DVG) and illumination cache (ICC). The integral equation in Eq. (2) was numerically solved for photon tracing (VPT). Two image resolutions (Image size) were tested where the ray marching time was almost proportional to the number of image pixels as indicated in the RM column. Except Cloud2, the adaptive, three-pass method was used for the density volume generation (DVG) on the GPU. For an adaptive density generation and photon gathering on the CPU, we implemented the kd-tree structure for nearest neighbor searching.*

	DVG	VPT	ICC	RM
$100 \times 54 \times 101$ (a)	0.11	0.54	0.69	0.54
$150 \times 80 \times 151$	0.15	0.51	1.21	0.54
$200 \times 107 \times 201$ (b)	0.23	0.49	2.60	0.58
$249 \times 133 \times 251$	0.35	0.53	4.67	0.62
$299 \times 160 \times 301$ (c)	0.55	0.74	8.29	0.66

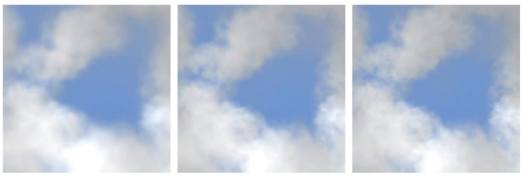


Figure 6: *Timing performance at multiple volume resolutions (in seconds). The run time for producing a 1024×576 image from the Cloud2 data was measured for different grid resolutions, holding all other rendering parameters fixed. A part of rendering results are compared for the three resolutions (a), (b), and (c), where the details became blurred as the grid resolution got lowered.*

permits an easy extension to various interesting rendering effects. Currently, we are including the light emission phenomena of hot gaseous fluids so that such important rendering effects as fire, flame, and explosion can be realistically and efficiently synthesized.

Acknowledgements: This work was supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2008-313-D00920).

References

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

[BNM*08] BOUTHORS A., NEYRET F., MAX N., BRUNETON E., CRASSIN C.: Interactive multiple anisotropic scattering in clouds. In *Proc. of ACM SIGGRAPH 2008 Symposium on Interactive 3D Graphics and Games* (2008), pp. 173–182.

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering. In *Proc. of ACM SIGGRAPH 2009 Symposium on Interactive 3D Graphics and Games* (2009), pp. 15–22.

[CPP*05] CEREZO E., PÉREZ F., PUEYO X., SERÓN F., SILLION X.: A survey on participating media rendering techniques. *The Visual Computer 21*, 5 (2005), 303–328.

[DYN02] DOBASHI Y., YAMAMOTO T., NISHITA T.: Interactive rendering of atmospheric scattering effects using graphics hardware. In *Proc. of Graphics Hardware 2002* (2002), pp. 99–107.

[EMPP03] EBERT D., MUSGRAVE F., PEACHEY D., PERLIN K.: *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 2003.

[HAP05] HEGEMAN K., ASHIKHMIN M., PREMOŽE S.: A lighting model for general participating media. In *Proc. of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games* (2005), pp. 117–124.

[HL01] HARRIS M., LASTRA A.: Real-time cloud rendering. In *Proc. of Eurographics 2001* (2001), pp. 76–84.

[JC98] JENSEN H. W., CHRISTENSEN P. H.: Efficient simulation of light transport in scenes with participating media using photon maps. In *Proc. of ACM SIGGRAPH 1998* (1998), pp. 311–320.

[JDZJ08] JAROSZ W., DONNER C., ZWICKER M., JENSEN H. W.: Radiance caching for participating media. *ACM Transactions on Graphics 27*, 1 (2008), Article No. 7.

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*. A K Peters, 2001.

[Kap03] KAPLER A.: Avalanche! snowy FX for XXX. In *Proc. of ACM SIGGRAPH 2003 Sketches & Applications* (2003), pp. 1–1.

[KH84] KAJIYA J., HERZEN B. V.: Ray tracing volume densities. In *Proc. of ACM SIGGRAPH 1984* (1984), pp. 165–174.

[KPh*03] KNISS J., PREMOŽE S., HANSEN C., SHIRLEY P., MCPHERSON A.: A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 150–162.

[Mon88] MONAGHAN J.: An introduction to SPH. *Computer Physics Communications 48*, 1 (1988), 89–96.

[PARN04] PREMOŽE S., ASHIKHMIN M., RAMAMOORTHI R., NAYAR S.: Practical rendering of multiple scattering effects in participating media. In *Proc. of Eurographics Symposium on Rendering 2004* (2004), pp. 363–374.

[PDC*03] PURCELL T., DONNER C., CAMMARANO M., JENSEN H., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *Proc. of the ACM SIGGRAPH/Eurographics Conference on Graphics Hardware* (2003), pp. 41–50.

[PKK00] PAULY M., KOLLIG T., KELLER A.: Metropolis light transport for participating media. In *Proc. of 11th Eurographics Workshop on Rendering 2000* (2000), pp. 11–22.

[QXF*07] QIU F., XU F., FAN Z., NEOPHYTOS N., KAUFMAN A., MUELLER K.: Lattice-based volumetric global illumination. *IEEE Transactions on Visualization and Computer Graphics*, 6 (2007), 1576–1583.

[Ree83] REEVES W.: Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics 2*, 2 (1983), 91–108.

[REK*04] RILEY K., EBERT D., KRAUS M., TESSENDORF J., HANSEN C.: Efficient rendering of atmospheric phenomena. In *Proc. of Eurographics Symposium on Rendering 2004* (2004), pp. 375–386.

[SKSU05] SZIRMAY-KALOS L., SBERT M., UMMENHOFFER T.: Real-time multiple scattering in participating media with illumination networks. In *Proc. of Eurographics Symposium on Rendering 2005* (2005), pp. 277–282.

[SRNN05] SUN B., RAMAMOORTHI R., NARASIMHAN S., NAYAR S.: A practical analytic single scattering model for real time rendering. *ACM Transactions on Graphics (ACM SIGGRAPH 2005)*, 3 (2005), 1040–1049.

[ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time kd-tree construction on graphics hardware. *ACM Transactions on Graphics 27*, 5 (2008), 1–11.

[ZRL*08] ZHOU K., REN Z., LIN S., BAO H., GUO B., SHUM H.: Real-time smoke rendering using compensated ray marching. *ACM Transactions on Graphics (ACM SIGGRAPH 2008) 27*, 3 (2008), Article No. 36.

Appendix A. Major rendering parameters for our particle volume renderer

Density Volume Generation

- $n_x \times n_y \times n_z$: 3D volume resolution (in the paper)
- $h^{(0)}$: initial value of smoothing length (in the paper)
- N^*: target particle number (in the paper)
- s: scale factor in relaxation coefficient α (in the paper)
- m_p: mass of particle p (in the paper)
- s_{denno}: density noise scale factor ($s_{denno} = 0$ if no noise is applied)

Volume Photon Tracing

- d_{MAX}: maximum distance to the next interaction (optional)
- Δx: step size used in numerical integration (in the paper)
- Λ: albedo of participating medium
- g: Henyey-Greenstein phase function parameter
- $s_{extcoef}$: scale factor for extinction coefficient ($\sigma_t(\boldsymbol{p}(x)) = s_{extcoef} \cdot d(\boldsymbol{p}(x))$, in the paper)
- N_{emph}: maximum number of photons emitted
- N_{MAXsto}: maximum number of photons stored
- N_{MAXint}: maximum number of photon interactions

Illumination Cache Construction

- $n_x \times n_y \times n_z$: 3D volume resolution (in the paper)
- h: photon gathering radius factor (in the paper)
- s_{phpow}: scale factor for photon power
- Λ: albedo of participating medium
- $s_{extcoef}$: scale factor for extinction coefficient ($\sigma_t(\boldsymbol{p}(x)) = s_{extcoef} \cdot d(\boldsymbol{p}(x))$, in the paper)
- g: Henyey-Greenstein phase function parameter
- $(\Delta x)'$: step size used for radiance sampling during single scattering computation

Ray Marching

- $n_w \times n_h$: image resolution
- $(\Delta x)''$: step size used for radiance sampling during final ray marching
- Λ: albedo of participating medium
- $s_{extcoef}$: scale factor for extinction coefficient ($\sigma_t(\boldsymbol{p}(x)) = s_{extcoef} \cdot d(\boldsymbol{p}(x))$, in the paper)
- s_{offno}: offset noise scale factor ($s_{offno} = 0$ if no noise is applied)