情報処理学会論文誌 Vol. 49 No. 5 1743–1749 (May 2008)

GPUシェーダを用いた動的な雲の高速レンダリング

嶋田有紀[†1] 新谷幹夫[†2]
白石路雄[†2] 原田隆宏[†3]

野外風景を写実的に描画するためには，高品質な雲の表現が欠かせない．雲の実時間レンダリング手法としてはHarrisの手法が代表的であるが，1) 光源方向の変化に対する処理時間が大きい，2) 雲の詳細な濃淡が表現できない，などの点で課題が残されている．本論文では，3次元テクスチャの利用や乗法型の輝度計算の導入によりHarrisの手法を改良し，高速かつ高品質な雲の画像生成法を提案する．本手法をピクセルシェーダで実装し，実験を行ったところ，従来実装と比べ10倍以上の高速化が実現された．また，生成画像の品質も大幅に向上した．

Fast Rendering of Dynamic Clouds Using GPU Shader

Yuki Shimada,[†1] Mikio Shinya,[†2] Michio Shiraishi[†2]
and Takahiro Harada[†3]

This paper presents a fast cloud rendering method for dynamic scenes, where cloud shapes and lighting environments dynamically change. Although the Harris method has been widely used for static cloud rendering, it can be fatally slow for real-time applications when, for example, light directions change. By introducing the 3D attenuation buffer and re-arranging the algorithm, we improved the rendering speeds of dynamic clouds by a factor of 10-100 times. The image quality was also improved due to a finer representation of light distribution.

†1 プロメテックソフトウェア
Prometec Software
†2 東邦大学
Toho University
†3 東京大学
The University of Tokyo

1. はじめに

野外風景を写実的に描画するためには，高品質な雲の表現が欠かせない．雲のレンダリングでは，雲内部における太陽光の散乱を計算する必要がある．散乱光レンダリングの研究は歴史が長く，82年にBlinn[1)]が一次散乱近似を導入した後，Kajiyaら[2)]が複数回散乱を考慮したボリュームデータのレイトレーシングを提案している．また，Nishitaら[3)]は異方性複数回散乱と天空光を考慮した雲の大局照明モデルとその実用的な実装方法を提案した．

2000年代に入ると，GPUを用いた実時間レンダリングが実現された．まず，Dobashiらはαブレンディングを用いることで雲による1次散乱を実時間表示できることを示した[4)]．さらに，HarrisらはDobashiらの手法をベースに，Phase functionの扱いなど実装上の工夫を施し，その詳細を明確に論じた[5),6)]．ソースコードの公開なども相まって，同手法は広く使用されるようになってきている．

Harrisの手法は，前計算工程とレンダリング工程の2つの工程からなる．前計算工程では，雲の各部位に到達する太陽光の輝度を計算し，レンダリング工程では，雲により視線方向に散乱する光量を視線方向に沿って積算することで雲のレンダリングを行う．しかし，前計算工程は，低速なGPU-CPUデータ転送（リードバック）の多用やパーティクルのソーティング処理などのため，処理速度が遅い．静的な情景では，前計算は1度だけで済むので影響が小さいが，光源位置や雲の密度などが変化する動的な情景では前計算を繰り返す必要があり，ゲームなどの実時間応用は困難となっている．一方，Miyazakiらは[7)]，shadow view sliceを導入することで前計算とレンダリング処理を統合し，動的な雲の描画を高速化を試みている．しかし，この処理では，雲の輝度計算を毎フレーム行うことになり，レンダリング処理がHarris法より低速になることは避けられない．太陽の位置移動や雲の移動による輝度の更新をすべてのフレームで常時行う必然性は低いので，必要に応じて前計算を高速に処理する方式が望ましいといえる．

そこで本論文では，Harris法の前計算工程をGPUプログラミングにより改良した手法を提案し，動的な雲の実時間描画を可能とする．具体的には，3次元テクスチャの利用や輝度計算方法の見直しにより，GPU-CPUリードバックおよびソーティング処理を排除し，前計算を高速化する．さらに，輝度分布をパーティクル単位ではなく，3次元テクスチャの画素単位で保持することにより，画像品質の向上も実現する．

2. Harris の手法

Harris 法のアルゴリズムは前計算工程とレンダリング工程の 2 つの工程からなる．前計算工程では，太陽からの光が雲内部の水滴による散乱・減衰を経て，雲の各部位に届く輝度を計算する．レンダリング工程では前計算工程で求めた雲の各部位の輝度を参照し，視線方向に散乱する量を視線方向に沿って積算することで雲のレンダリングを行う．雲内部に存在する水分の分布は，**図 1** に示すように，ガウス分布に従う濃度値のテクスチャが張られたポリゴン（パーティクル）の集合で表現する．

前計算およびレンダリングの処理は，このパーティクル単位で行う．以下に述べるように，前処理はレンダリング処理に比べて非常に低速である．雲形状や光源方向の変化を実時間表示するには，前処理の改良が必要不可欠である．

2.1 前計算工程

前計算工程では，光源から入射して各雲パーティクルに届く輝度を求める．本来，散乱現象を正確にシミュレートするには各パーティクルについて全方向からの入射と全方向への散乱を考慮する必要があるが，Harris 法では計算コストを抑えるため，入射方向における光の吸収と散乱のみを考慮する．各パーティクルに届く輝度 I_P は以下のように計算される．

$$I_P = I_0 \prod_{j=1}^{N} e^{-\tau_j} + \sum_{j=1}^{N} g_j \prod_{k=j+1}^{N} e^{-\tau_k} \tag{1}$$

$$g_k = a_k \tau_k p(l', l) I_k / 4\pi \tag{2}$$

ここで，I_0 は雲に入射する直前の太陽光の輝度であり，N はパーティクル P より光源に近いパーティクルの数である．τ_k は減衰率，a_k はアルベド（減衰率のうち散乱が占める割合）である．$e^{-\tau_j}$ はパーティクルの透明度を示す．**図 2** に示すように，式 (1) は各パーティクルに届く輝度 I_p が，

- パーティクル P に散乱せずに到達した光の輝度 $I_0 \prod_{j=1}^{N} e^{-\tau_j}$
- パーティクル j からパーティクル P に向かって散乱した光の輝度の和 $\sum_{j=1}^{N} g_j \prod_{k=j+1}^{N} e^{-\tau_k}$

の 2 つを足し合わせたものであることを意味している．式 (2) の関数 $p(l', l)$ は phase function と呼ばれ，l 方角から入射した光が，l' 方角にどれだけの割合で散乱するかを定義する関数である．雲の散乱に適した phase function としては Mie phase function などが知られている[8]．本来は全方向についての入射と散乱を考える必要があるが，計算コストを抑えるため，式 (2) において入射方向と放射方向をともに l としている．よって $p(l, l)$ は定数値になる．

この理論式を GPU に合わせて漸化式として表現すると以下のようになる．

$$I_k = \begin{cases} g_{k-1} + T_{k-1} \cdot I_{k-1} & (2 \le k \le N) \\ I_0 & (k = 1) \end{cases} \tag{3}$$

$$g_{k-1} = a_{k-1} \cdot \tau_{k-1} \cdot p(l, l) \cdot I_{k-1} / 4\pi \tag{4}$$

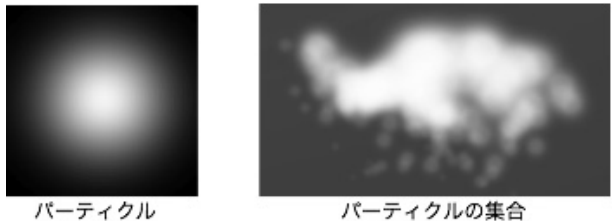

図 1　パーティクルによる雲の水分分布の表現
Fig. 1　Particle representation of water distribution in clouds.

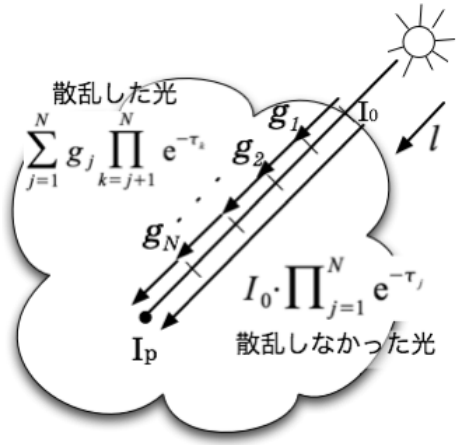

図 2　Harris 法における前計算工程
Fig. 2　Pre-process in the Harris method.

ここで T_k はパーティクル k の透明度を表している．

この計算はパーティクルをソートし，太陽に近い順に行う．I_k は k 番目のパーティクルに届いた光源からの輝度であり，GPU のフレームバッファの値として保持される．式 (3) は GPU のアルファブレンディングによって計算できる．バッファは上書きされてしまうので，パーティクルの輝度 I_k はそのつどリードバックし，主記憶上に保持される．これは，レンダリング工程においてすべての I_k が必要なためである．現在の GPU はリードバックが非常に低速なので，この部分が前計算工程の最大の速度低下要因となっている．また，GPU に比べて低速な CPU において，式 (4) の計算とパーティクルのソート作業を行っている．これも速度が低下する要因となっている．これ以外にも，輝度がパーティクルごとに記憶されるので，輝度の精細な変化を表現できないという問題がある．

2.2　レンダリング工程

式 (1) と同様に，レンダリングの理論式を離散化すると次の式を得る．この式の処理は視点から遠い順に行う．

$$E_k = S_k + T_k \cdot E_{k-1} \quad (1 \le k \le N) \tag{5}$$

$$S_k = a_k \cdot \tau_k \cdot p(\omega, l) \cdot I_k / 4\pi \tag{6}$$

ここで，E_k はパーティクル k における視線方向の輝度である．E_k は光源からパーティクル k に届いた光（I_k）が視点方向に散乱する輝度 S_k と，1 つ前のパーティクル $k-1$ からの輝度 E_{k-1} が T_k だけ減衰した量 $T_k E_{k-1}$ の和で計算できる．I_k は前計算で計算しており，T_k はパーティクル k の透明度，ω は視線方向である．この式は GPU のアルファブレンディングで計算でき，処理速度上の問題はない．

3. 提 案 手 法

3.1　改良の概要

提案手法では前計算工程を改良し，以下のような改善を実現している．

(1)　低速なピクセルリードバック処理が必要ない．
(2)　式 (3)，式 (4) の計算すべてを高速な GPU で行える．
(3)　前計算工程で，パーティクルのソートが必要ない．
(4)　CPU はすべてのパーティクルを 1 度に送信でき（Vertex Buffer Object），GPU は高効率にパイプライン計算できる．
(5)　雲の各部位に到達する輝度を，パーティクル単位よりも細かい，3D テクスチャのボクセル単位で処理する．これにより，画像品質が向上する．

我々の提案手法では，上記の (1)，(2)，(3) を実現するために**乗算型の輝度計算**を，(4)，(5) を実現するために **3D 輝度空間**を導入する．これらの詳細については後述する．

以下，提案実装の詳細について説明する．

3.2　リードバック排除と GPU での散乱漸次式の計算

まず，一番のボトルネックであるピクセルリードバックを考察する．I_{k-1} の値をリードバックする理由は，1) CPU 側で I_{k-1} を用いて式 (4) を計算する，2) レンダリング時に CPU 側で値を参照する，である．したがって，1) GPU プログラムで式 (3)，(4) を計算する，2) 値を GPU 側のテクスチャメモリに格納し，レンダリング時に参照する，ことによりリードバックを回避できる．このうち，実装上問題となるのは 1) である．まず，式 (3)，式 (4) を再び示すと，

$$I_k = \begin{cases} g_{k-1} + T_{k-1} \cdot I_{k-1} & (2 \le k \le N) \\ I_0 & (k = 1) \end{cases}$$

$$g_{k-1} = a_{k-1} \cdot \tau_{k-1} \cdot p(l, l) \cdot I_{k-1} / 4\pi$$

となる．これを GPU でレンダリングするには，a) I_{k-1} を処理中のバッファ（レンダターゲットバッファ）から参照，b) 式 (4) を計算，c) α ブレンディングにより積算して I_k をレンダターゲットに書き込む，という処理を行うのが自然である．しかし，現在の GPU ではレンダターゲットバッファの値を同時にピクセルシェーダから参照した場合の動作は保証されていないので，処理 a) が実装できない．したがって，I_{k-1} を参照しない計算式により I_k を求めることが実装上必要とされる．そこで，式 (3)，(4) を変形し，

$$\begin{aligned} I_k &= G_k I_{k-1} = \left(\prod G_i \right) I_0 \\ G_i &= (a_{k-1} \cdot \tau_{k-1} \cdot p(l, l)/4\pi + T_{k-1}) \\ \Gamma &= \prod G_i \end{aligned} \tag{7}$$

とする．I_k，I_{k-1} をレンダターゲットバッファの値と考えると，この式では，レンダターゲットを参照していないので，単に G_i を積算するだけで GPU 実装が可能となる．これが前節で述べた乗算型の輝度計算である．また，この方法ではパーティクルをソーティングする必要もなくなる．すなわち，式 (7) の右辺の ($\Gamma = \prod G_i$) は単なるスカラー乗算なので，処理の順番には依存しないからである．以上の改良で，3.1 節であげた提案手法の利点の (1)，(2)，(3) に関しては実現できた．

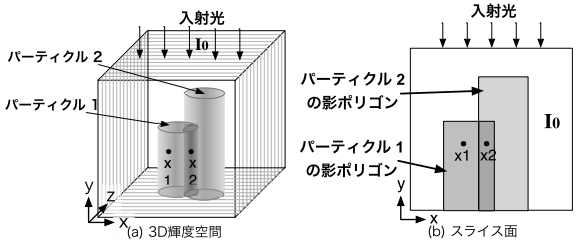

図 3　3D 輝度空間
Fig. 3　3D intensity space.

3.3　3D 輝度空間による計算速度と画像品質の向上

式 (7) の計算では，Γ 値を保持，更新する必要がある．そこで 3.1 節で述べた 3D 輝度空間を導入する．3D 輝度空間は描画対象となる雲を内包する 3D 空間であり，ボクセルごとに，Γ 値を保存する．

図 3 (a) に示すように，3D 輝度空間の xz 平面を光源方向と垂直になるように設定する．また，z 軸に垂直なボクセル平面をスライス面と呼ぶことにする．

Γ 値の計算を図 3 (a) に示す例で説明する．パーティクル 1 が Γ 値に影響を与える領域は，図に示す円筒領域である．円筒のスライス面での断面は図 3 (b) に示すように，長方形となる．これを影ポリゴンと呼ぶことにする．x1 はパーティクル 1 の影ポリゴン内にあり，その Γ 値は G_1 となる．一方 x2 はパーティクル 1 と 2 双方の影ポリゴン内にあるので，Γ 値は $G_1 * G_2$ となる．すなわち，影ポリゴンを G_i で塗りつぶし，乗算ブレンディングすれば，各ボクセルにおける Γ 値が計算できる．レンダリング工程では，各ボクセルに格納された Γ 値を参照し，I_0 を掛けることでその雲の部位の輝度が求まる．

従来手法では輝度の保存がパーティクルごとという荒い単位であったが，提案手法では 3D 輝度空間のボクセルごとに保存される．このため，レンダリング品質も向上する．

4.　提案手法の実装

4.1　頂点シェーダでの処理

3D 輝度空間における，影ポリゴンの各頂点は以下のように設定する．

- 左上頂点：$(x_p - r_p, y_p + r_p, z_i)$
- 左下頂点：$(x_p - r_p, 0, z_i)$
- 右上頂点：$(x_p + r_p, y_p + r_p, z_i)$
- 右下頂点：$(x_p + r_p, 0, z_i)$

ただし，(x_p, y_p, z_p) はパーティクル中心の位置，r_p はパーティクルの半径，z_i は交差したスライス面の z 座標である．図 3 (b) に示すように，このポリゴンを z 方向から見ると，長方形の形状となる．

以上から，影の描画には以下の 3 つの処理が必要であることが分かる．

A)　各パーティクルについて，交差するスライス面とその面数 n_p を求める．
B)　各影ポリゴンについて，頂点座標を計算する．
C)　影ポリゴンを送信する．

処理 A）と B）は CPU よりも GPU の頂点シェーダで実行した方が高速である．各パーティクルが必要とする影ポリゴンの数 n_p は一定ではないが，2007 年現在で最も普及している DirectX9 世代の GPU では，GPU 内部でポリゴン数を増減することはできない．そこで以下の方法をとった．

必要となる影ポリゴンの最大数 n_{pmax} は最大のパーティクル半径によって事前に決めることができる．そこで，すべてのパーティクルに対して n_{pmax} 個の影ポリゴンを GPU に送信する．多くの場合，各パーティクルの前計算処理に必要な実際のポリゴン数 n_p は n_{pmax} より少ない．そこで頂点シェーダの B）の計算において，不必要な $n_{\mathrm{pmax}} - n_p$ 個のポリゴンは全頂点を同一座標とし，縮退させる．こうすることでこれらのポリゴンはラスタライズ以降の処理が行われなくなる．以上の頂点シェーダ内の処理は，動的分岐をいっさい使わずに実装することができる．

処理 C）に関しては，送るべきポリゴン数があらかじめ決まっているため，頂点配列や VBO，ディスプレイリストなどを用いて高速に転送することができる．また，現在の GPU では仕様上の制限から，3D テクスチャへの任意の位置への書き込みは行えない．そこで実際の実装では，出力先の 3D 輝度空間を 3D テクスチャではなく，各スライス面に対応するタイルをラスタ順に敷き詰めた大きな 2D テクスチャとして表現する．以後，これを輝度タイルテクスチャと呼ぶ．そこで頂点シェーダにおいて，処理 B）の座標計算で得られた 3 次元テクスチャ座標を**図 4** のように 2 次元座標に変換し，ピクセルシェーダで描画している．

4.2　ピクセルシェーダにおける処理

パーティクルの持つ水分はガウス分布に従っているので，ガウス分布テクスチャを参照し，これをパーティクルの透過率を乗ずることで各ピクセルでの透過率 G を求める．

式 (7) に示される乗算ブレンディングは現状の GPU ではサポートされていない．そこで，$\log(G)$ の値を描画し，加算ブレンディングを行う．レンダリング工程では，この exp

をとって輝度を求める．

4.3　3D テクスチャへのコピー

レンダリング工程の GPU フラグメント処理において，パーティクルのフラグメントは前計算で作成した 3D 輝度空間の輝度データにアクセスする．ここで 3D 輝度空間が 3D テクスチャとして表現されていれば，パーティクルビルボードのフラグメントごとにテクスチャフェッチを行い，さらに x，y，z 各方向について線形フィルタリングされたテクスチャ参照値を利用できるため，従来実装以上のレンダリング品質を得ることができる．そこで，輝度タイルテクスチャから 3D テクスチャへ輝度データのコピーを行う．

コピーの方法には次の 2 つのやり方がある．

(1)　glCopyTexSubImage3D 関数を使う方法

(2)　Framebuffer Object を使う方法

(1) の方法では，glCopyTexSubImage3D 関数を使い輝度タイルテクスチャの各タイル領域のピクセル値を 3D テクスチャの各 z 軸 2D スライスにコピーする．コピーの際，一部の GPU では内部ピクセル形式の違いから変換処理が入り，希望どおりのパフォーマンスが出ない恐れがあるため，様々なテクスチャ形式を試し一番速い組合せを見つける必要がある．

(2) の方法は Geforce8800 や一部の最新の Radeon でのみ行うことができる．3D テクスチャの 2D スライスを Framebuffer Object にアタッチし，レンダリング対象とする．2D 輝度タイルテクスチャの各タイル領域をポリゴンにテクスチャマップし，この 2D スライスに描画する．これは (1) の方法よりも高速にコピーを行うことができる．本実装では (2) の Framebuffer Object を利用する方法を用いた．

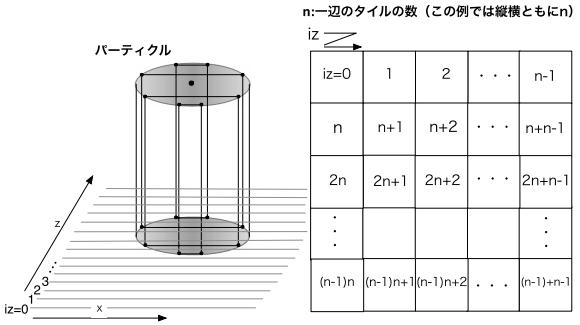

図 4　輝度タイルテクスチャ．n は縦横 1 辺あたりのタイル数．zsize は 3D 輝度空間の z 方向の解像度

Fig. 4　Intensity tiled texture. The number of tiles in each direction and the number of pixel planes in the z-direction is n and zsize.

5.　速度・品質比較

Geforce8800GTX，Intel Core 2 Duo1.83 GHz の環境で，前計算の処理速度を比較した．

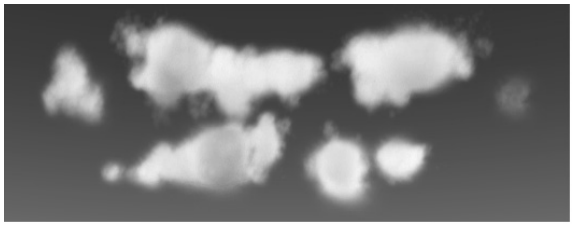

図 5　生成した 9382 パーティクルの雲

Fig. 5　Cloud with 9382 particles.

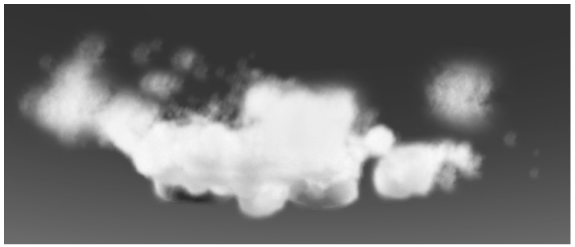

図 6　生成した 32914 パーティクルの雲

Fig. 6　Cloud with 32914 particles.

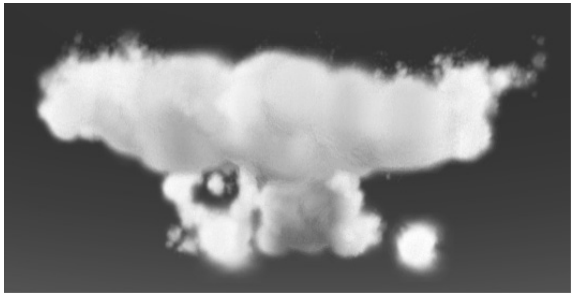

図 7　生成した 60134 パーティクルの雲

Fig. 7　Cloud with 60134 particles.

表 1　処理速度の比較（提案手法のカッコ内は 3D テクスチャの解像度）
Table 1　Comparison of the processing speed.

	9382 particles	32914 particles	60134 particles
Harris 従来手法	0.721 sec	3.164 sec	5.934 sec
提案手法 (384^3)	0.038	0.154	0.161
提案手法 (256^3)	0.015	0.044	0.055
提案手法 (128^3)	0.007	0.015	0.016

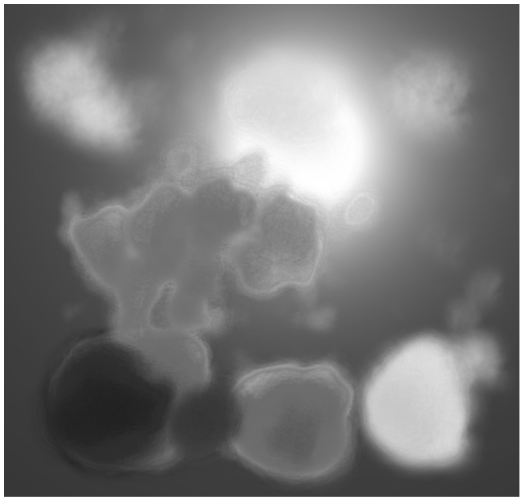

図 8　従来実装による生成画像例
Fig. 8　Example image synthesized by the previous method.

得られた画像を**図 5**，**図 6**，**図 7** に示し，その結果を**表 1** に示す．

表に示されるように，8800GTX では従来手法と比べ 10〜100 倍以上の高速化が得られることが判明した．

また，提案手法は従来手法と比べ画像品質においても大幅な向上を実現している（**図 8**，**図 9**）．これは，提案実装では輝度の保存と参照を 3 次元のボクセル単位でより細かく行っているためである．

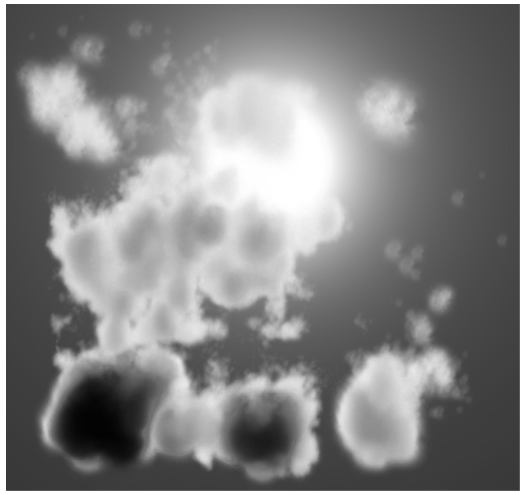

図 9　提案手法による生成画像例
Fig. 9　Example image synthesized by the proposed method.

6. ま と め

本論文では Harris の手法のより高速，高品質な実装方法を提案した．本手法により，光の散乱を考慮した高品質な雲の表示を，太陽位置や雲データが変化する条件下でもリアルタイムに更新することが可能となった．

今回の実装では，DirectX9 世代の GPU を想定したが，DirectX10 世代の GPU では GPU 内部でポリゴンの増減が可能になるので，前計算において CPU 側が GPU に送るべきポリゴンの数を動的に制御し，CPU・GPU 間の帯域の消費を抑えることができる．これにより，3D 輝度空間の解像度やパーティクルのサイズが大きい場合には大きな効率化が期待できる．また，今後の課題として，多重散乱効果の導入なども検討してゆきたい．

参 考 文 献

1) Blinn, J.: Light Reflection Functions for Simulation of Clouds and Dusty Surfaces, *SIGGRAPH 1982*, pp.21–29 (1982).
2) Kajiya, J. and Von Herzen, B.: Ray Tracing Volume Densities, *SIGGRAPH 1984*,

pp.165–174 (1984).
3) Nishita, T., Dobashi, Y. and Nakamae, E.: Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light, *SIGGRAPH 1996*, pp.379–386 (1996).
4) Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T. and Nishita, T.: A Simple, Efficient Method for Realistic Animation of Clouds, *SIGGRAPH 2000*, pp.19–28 (2000).
5) Harris, M.: Real-Time Cloud Rendering for Games, *Proc. Game Developers Conference 2002* (2002).
6) Harris, M. and Lastra, A.: Real-Time Cloud Rendering, *Computer Graphics Forum* (*Proc. Eurographics 2001*), Vol.20, No.3, pp.76–84 (2001).
7) Miyazaki, R., Dobashi, Y. and Nishita, T.: A Fast Rendering Method of Clouds using Shadow-View Slices, *Proc. CGIM 2004*, pp.93–98 (2004).
8) Preetham, A.: Modeling Skylight and Aerial Perspective, *Course Note at SIGGRAPH 2003* (2003).
9) Reeves, W.: Particle Systems – A Technique for Modeling a Class of Fuzzy Objects, *ACM Trans. Graphics*, Vol.2, No.2 (1983).
10) Wang, N.: Realistic and Fast Cloud Rendering, *sketch at SIGGRAPH 2003* (2003).

(平成 19 年 8 月 2 日受付)
(平成 20 年 2 月 5 日採録)

嶋田　有紀

昭和 57 年生．平成 19 年東邦大学大学院理学研究科情報科学専攻修士課程修了．同年プロメテック・ソフトウェア（株）入社．ゲームエンジン・物理エンジンの研究開発に従事．

新谷　幹夫（正会員）

昭和 54 年早稲田大学理工学部応用物理学科卒業．昭和 56 年同大学院理工学研究科物理および応用物理学専攻修士課程修了．同年日本電信電話公社武蔵野電気通信研究所入所．視覚系の心理物理学的研究，文字認識の研究，コンピュータグラフィックスの研究等に従事．平成 13 年東邦大学理学部情報科学科教授．工学博士．シミュレーション外科学会，画像電子学会，電子情報通信学会，ACM 各会員．

白石　路雄（正会員）

昭和 49 年生．平成 15 年東京大学大学院総合文化研究科広域科学専攻博士課程修了．博士（学術）．同年株式会社ガリレオ入社．現在，東邦大学理学部情報科学科専任講師．コンピュータグラフィックス，特にノンフォトリアリスティックレンダリングの研究に従事．画像電子学会，ACM，IEEE-CS 各会員．

原田　隆宏（正会員）

昭和 56 年生．平成 18 年東京大学大学院工学系研究科システム量子工学専攻修士課程修了．同年東京大学大学院情報学環助手．平成 19 年 4 月助教．計算力学とコンピュータグラフィックスの研究に従事．平成 19 年度情報処理学会山下記念研究賞．日本機械学会，画像電子学会，ACM 各会員．