Fast Rendering of Dynamic Clouds

Horng-Shyang Liao [a,b,*] Tan-Chi Ho [b,**] Jung-Hong Chuang [b,**]
Cheng-Chung Lin [b,**]

[a] *National Center for High-Performance Computing*
Taiwan, R.O.C.

[b] *Department of Computer Science and Information Engineering*
National Chiao Tung University, Taiwan, R.O.C.

Abstract

We propose an efficient framework for dynamic cloud rendering. Base on the proposed simplified lighting model, the *Shadow Relation Table (SRT)* and *Metaball Lighting Texture Database (MLTDB)* are constructed in the preprocessing, and features such as self-shadowing and light scattering can be computed quickly using table lookup at the run time. Clouds are rendered using textured billboards and alpha blending, which is further speeded up by incorporating the octree hierarchy and the hierarchical texture caching.

Key words: cloud simulation, dynamic clouds rendering, cellular automata

1 Introduction

Dynamic and photorealistic rendering of cloud is an indispensable element to flight simulation, outdoor scenes, and many other virtual environments. Due to the high computational cost for simulation and rendering, lots of research efforts have been devoted to reduce the computation time, which can be roughly divided into three categories:

- Simplifying physics of simulation and modeling (1; 2; 3; 4; 5; 6; 7; 8; 9)
- Photorealistic or special effects rendering (3; 10; 11; 12; 13; 14; 15)
- Efficient or real-time rendering (2; 16; 17; 18)

* c00hxl00@nchc.org.tw
** {danki, jhchuang, cclin}@csie.nctu.edu.tw

Recently, the real-time rendering of dynamic cloud has become a major issue, and an attempt of ours to such a goal is presented here.

The process of photorealistic dynamic cloud rendering generally consists of three components: simulation, modeling, and rendering. For simulation, cellular automata proposed by Dobashi et al. (2) is employed here due to its relatively lower computational cost. We modify some of the original simulation rules in cellular automata to be suited for run-time simulation. Note that, due to the limitation of cellular automata, we deal with only the animation of cumulus-like clouds. The proposed framework is specially engineered in the modeling and rendering stages with the following features:

- A simplified lighting model is devised, establishing the *Shadow Relation Table (SRT)* and the *Metaball Lighting Texture Database (MLTDB)* in preprocessing stage to speed up the run-time rendering by reducing the run-time illumination computation to a table look-up.
- An octree representation is constructed for the simulation volume as the basis for supporting back-to-front traversal, view frustum culling, hierarchical texture caching, and other operations necessary for real-time rendering.
- A hierarchical texture caching that aims to reduce the texture selection cost and the number of billboards.

The proposed rendering system is implemented on a dual-CPUs PC, taking into account the load balance between each CPU and graphics hardware. The experiment reveals that the proposed method renders visually convincing results with encouraging FPS ranging from 9 to 60 depending on the location of camera, the resolution of simulation volume, and the threshold values of hierarchical texture caching.

2 Related Work

In this section, We briefly review the related works of cloud rendering, regarding the simulation, modeling, and rendering.

SIMULATION: Depending on the application, the approaches to cloud simulation can either be physics-based (3; 4; 5; 7; 8; 19) or be heuristic (1; 2; 11; 12; 13).

Physics-based cloud simulation has to deal with fluid dynamics by solving Navier-Stokes equations, which is computationally expensive. Simplified versions, such as Stable Fluid (7) and coupled map lattice (CML) (5), that discard components of little visual effects have been proposed for graphics applications. The former does not deal with vapor-water transition and thus does not fit for cloud simulation but for smoke, the latter is suitable for clouds. However, both are time consuming.

Heuristic approaches, most of which being procedural modeling, are somewhat inexpensive in computation, but not so "user friendly" and have to be tuned by trial-and-error. The cellular automata proposed by Dobashi et al. (2) is perhaps an exception because it is easy to use and efficient.

Harris et al. simulate the cloud dynamics using programmable graphics hardware (20). The cloud simulation based on partial differential equations are developed, and a staggered grid discritization is used to perform the advection of simulation. All the computations are performed on the graphics hardware.

MODELING: Clouds can not be modeled as geometric entities, but as density distribution of some sorts. The treatments of which can be classified as particle system (6; 17), volume with metaball (2; 12; 19), and image based modeling (21). Particle system may capture the dynamics of gaseous effects well, but would become cumbersome when highly detailed rendering results are required. Metaball is a mathematical model for describing the density field, and has been widely used for representing volume density of cloud. The density of a point in the simulation space is just the sum of all density values of the point computed from all metaballs covering the point. Image based modeling renders clouds on imagery basis.

RENDERING: The mostly concerned issue of cloud rendering is the photorealistic quality, for which the scattering and absorbing effects within clouds must be taken care of (10). Ray tracing (15) and its extension — photon map (14), and procedural texturing (11; 12; 13; 22) are common techniques for such purposes, but all time consuming. Some approaches trade image quality for rendering efficiency (2; 17), in which the rendering is in general split into two passes. The first pass calculates shadow or illumination and the second pass renders the final image. Dobashi et al. render dynamic clouds by creating shadow texture for each metaball from sun in the first pass and projecting voxels using alpha blending and texture mapping in the second pass (2). Harris and Lastra develop real-time static clouds rendering with precomputing illumination in first pass and updating impostor using Schaufler's method (23) in second pass (17).

3 System Overview

The proposed rendering system for dynamic clouds consists of the preprocessing and the run-time stage. Table 1 shows the tasks of simulation, modeling and rendering involved in preprocessing and run-time phase.

The simulation volume is constructed by the octree hierarchy. The preprocessing stage initiates the settings of cellular automata, and constructs MLTDB and SRT,

which will be used for speeding up the two-pass rendering at the run-time stage.

The run-time stage starts the running of cellular automata, which is used to determine the density distribution over time for each voxel within the simulation volume. Then the density information is smoothed into each voxel by a horizontal filter. A two-pass rendering method is used for cloud rendering. First, the illumination of the voxels are computed using SRT. And then the voxels are rendered using textured billboards in a back-to-front order. To further speed up the rendering, a hierarchical texture caching mechanism is used to reduce the number of billboards rendered.

Table 1
System overview.

	Simulation	Modeling	Rendering
Preprocessing	Initial setting (Octree)	Volume partition	MLTDB and SRT construction
Rum time	Clouds transition rules of cellular automata	Smooth density	Two-pass rendering and hierarchical texture caching.

4 Preprocessing

4.1 Clouds Simulation Initialization

Before running the cellular automata, the initial distribution of the cloud and vapor should be determined. The phase transition from vapor to water, as well as the probability associated with the parameters need also to be initialized. These probability distributions are controlled by metaball (24) and affect the macrostructure of clouds. In a metaball, vapor and phase transition probabilities are set higher in central portions than in the boundary regions and cloud extinction probability is set higher in the boundary regions than in the central portions so that the clouds would appear thinner in the surroundings and thicker in the middle.

Instead of using ellipsoids for controlling cloud motion as was done by Dobashi et al. (2), metaball function is employed in our system via the effects resulting from the use of horizontal filter at run-time for smoothing the density distribution.

4.2 MLTDB and SRT

4.2.1 Lighting Model

The lighting model involves the computation of shadow and scattering illumination for each voxel. For a voxel, let I_{in} be the input illumination of the voxel and I_{out} be the output illumination through the voxel. The value of I_{out} is derived from

$$I_{out} = I_{in}(1.0 - D_{voxel}\sigma_a\tau), \tag{1}$$

where D_{voxel} is the voxel density, σ_a is the illumination attenuation ratio(IAR) to control the attenuation of illumination, and τ is the thickness of cloud. The computation of Eq. 1 is inexpensive but critically depends on I_{in}, of which the evaluation will be detailed in Sec. 4.2.2.

The scattering illumination into eye is considered to be the composite effects of the scattering and the cloud transparency. It is described as follows:

$$B = \begin{cases} I_{in}\left\{\frac{1}{2}a\left[1+M\cos\left(\alpha_{L,E}\right)\right]+S\right\}D_{cld}, & \alpha_{L,E} \le \frac{\pi}{2} \\ I_{in}\left\{\frac{1}{2}a\left[1+M\cos\left(\alpha_{L,E}\right)\right]+S\right\}D_{cld} & \alpha_{L,E} > \frac{\pi}{2} \\ \qquad +M\cos\left(\pi-\alpha_{L,E}\right)I_{out}D_{cld}, & \end{cases} \tag{2}$$

where B is the illumination scattered into eye, $\alpha_{L,E}$ is the angle between the light ray and the eye ray, D_{cld} is the cloud density, a is the albedo, M is the material, and S is a constant value to approximate the multi-scattering effects. Note that the cloud density D_{cld} in a voxel is the result of multiplying the voxel density D_{voxel} by the projected metaball function, which is the projection of the metaball function described in (24) into a 2D plane. In Eq. 2, $\frac{1}{2}a(1+M\cos(\alpha_{L,E}))$ is the phase function suggested by Blinn (10). For $\alpha_{L,E}$ less or equal to 90°, the scattering illumination consists of only the scattering, but for $\alpha_{L,E}$ greater than 90°, the light transmits through the cloud should also be considered.

With the assumption that the distribution of the multi-scattering is similar to the single scattering, the S in Eq. 2 is replaced by a constant σ_{s1}, and another constant σ_{s2} is multiplied to the Eq. 2 to increase the brightness of clouds. The two constant values σ_{s1} and σ_{s2} are called *pseudo scattering coefficients (PSC)*. The Eq. 2 can then be rewritten as:

$$B = \begin{cases} I_{in}\left\{\frac{1}{2}a\left[1+M\cos\left(\alpha_{L,E}\right)\right]+\sigma_{s1}\right\}\sigma_{s2}D_{cld}, & \alpha_{L,E} \le \frac{\pi}{2} \\ I_{in}\left\{\frac{1}{2}a\left[1+M\cos\left(\alpha_{L,E}\right)\right]+\sigma_{s1}\right\}\sigma_{s2}D_{cld} & \alpha_{L,E} > \frac{\pi}{2} \\ \qquad +M\cos\left(\pi-\alpha_{L,E}\right)I_{out}\sigma_{s2}D_{cld}. & \end{cases} \tag{3}$$

4.2.2 Shadow Relation Table (SRT)

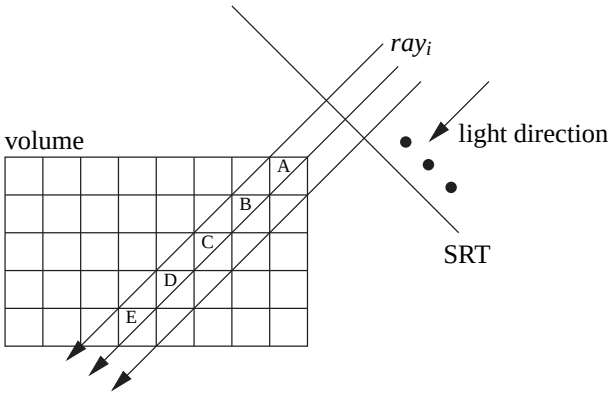


Fig. 1. Shadow relation table.

With the assumption that the sun remains still during simulation, we can link voxels using a table of 2D grid-structure, called *shadow relation table (SRT)*. SRT is placed between the sun and the simulation volume and has each of its entries associated with a linked list linking voxels intersected by the ray of sun beam in ascending order of distance from the sun. SRT is established as follows:

(1) Determine the orientation of incident ray of the sun beam.
(2) Place SRT between the sun and the simulation volume.
(3) Shoot enough rays from SRT grids to the simulation volumes in the orientation determined in step 1 until all voxels are covered.
(4) Cascade voxles on the same ray in ascending order by distance from the sun.

Fig. 1 shows the concept of SRT. Voxels A, B, C, D, and E are linked by ray_i in order.

It's possible that a voxel is intersected by more than one ray, as shown in Fig. 2, which will yield the incorrect illumination of the voxel. For such a voxel, it is reassigned to the ray with the largest value of $\overrightarrow{PC_v} \cdot L$, where c_v is the voxel's center.

With the aid of SRT, the input illumination I_{in} for every voxel can be determined very quickly, which is the output illumination I_{out} of the previous voxel in the linked-list.

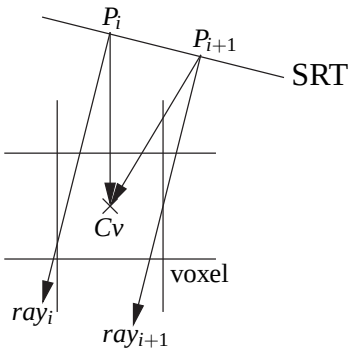


Fig. 2. Choosing a suitable ray.

4.2.3 Metaball Lighting Texture Database (MLTDB)

With I_{out} substituted by Eq. 1, the scattering illumination in Eq. 3 can be rewritten as

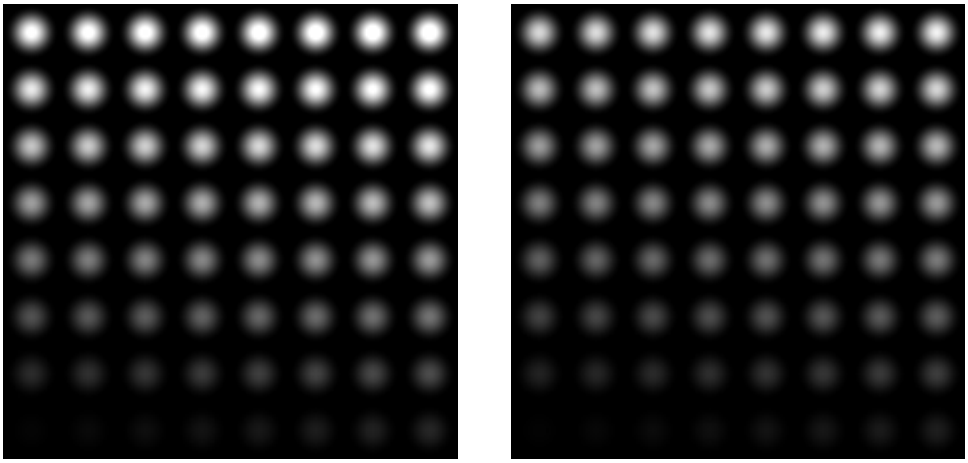
$$B = \begin{cases} I_{in}\left\{\frac{1}{2}a\left[1+M\cos\left(\alpha_{L,E}\right)\right]+\sigma_{s1}\right\}\sigma_{s2}D_{cld}, & \alpha_{L,E} \leq \frac{\pi}{2} \\ I_{in}\left\{\frac{1}{2}a\left[1+M\cos\left(\alpha_{L,E}\right)\right]+\sigma_{s1}\right. & \alpha_{L,E} > \frac{\pi}{2} \\ \qquad \left. +M\cos\left(\pi-\alpha_{L,E}\right)\left(1.0-D_{voxel}\sigma_a\tau\right)\right\}\sigma_{s2}D_{cld}. & \end{cases} \tag{4}$$

In Eq. 4, B is to be determined using parameters including I_{in}, D_{cld}, D_{voxel}, and $\alpha_{L,E}$ derived at run-time. Since I_{in} can be calculated quickly using SRT and Eq. 1, rendering speed-up would be possible if the major illumination calculations can be replaced by a table look-up. This is achieved by the proposed MLTDB, which is a database of 32×32 metaball textures varying in 64 voxel densities and viewed from 37 aspects. Each metaball texture stores the scattering illumination of the cloud in a voxel into eye computed using Eq. 4 without multiplying by I_{in}.

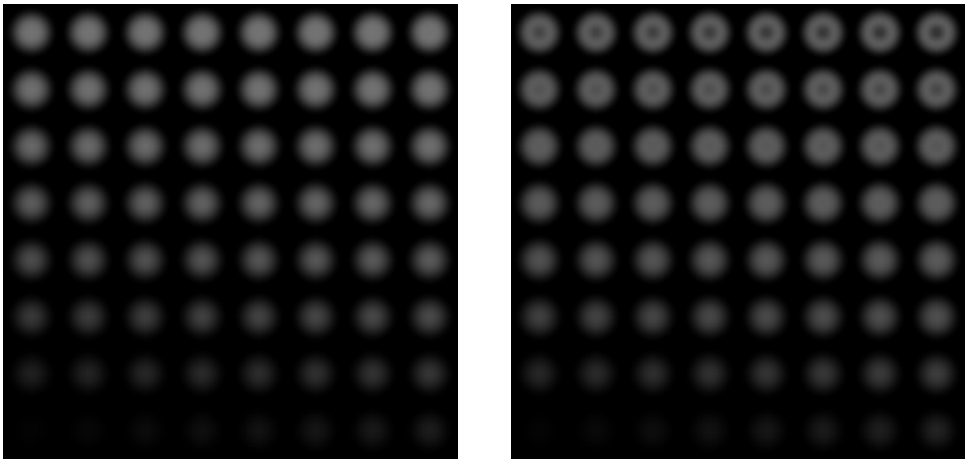
Using D_{voxel} and $\alpha_{L,E}$ as indices to the MLTDB, we can get a metaball texture, which is then multiplied by I_{in} to compute the scattering illumination B. The resultant scattering illumination is later used as the texture that associated with the billboard of the voxel for rendering. Fig. 3 depicts metaball textures selected from MLTDB, in which 64 textures are listed from the left to right and from bottom to the top for increasing $\alpha_{L,E}$.

5 Run Time Process

Fig. 4 illustrates the detail flowchart of the run time processing. Cellular automata is used for cloud simulation. The rendering is handled by a two-pass method based on a hierarchical octree structure. In the first pass, shadows are calculated using SRT. The second pass renders the final image using textured billboards in a back-to-front order.



(a) $\alpha_{L,E} = 0^\circ$ (b) $\alpha_{L,E} = 60^\circ$



(c) $\alpha_{L,E} = 120^\circ$ (d) $\alpha_{L,E} = 180^\circ$

Fig. 3. Selected images from MLTDB.

5.1 Simulation Using Cellular Automata

Cellular automata for clouds simulation was proposed by Dobashi et al. (2). It extends Nagal and Raschka's approach (25) in several aspects by considering the extinction of clouds, wind effects, faster simulation using bit-field manipulation and cloud motion control by using ellipsoids. We modify the cellular automata for run-time simulation by adding the following features:

- We consider the extinction ability of clouds in (1), and add the cloud-to-vapor probability to transform the clouds to vapor rather than just disappear.
- We make the constraint that the cloud growing and disappearing occur only in boundary regions.
- We reduce the growing ability of clouds to have the balance between it and the extinction ability.
- We add upper and lower bound for the number of cloud voxels and vapor voxels

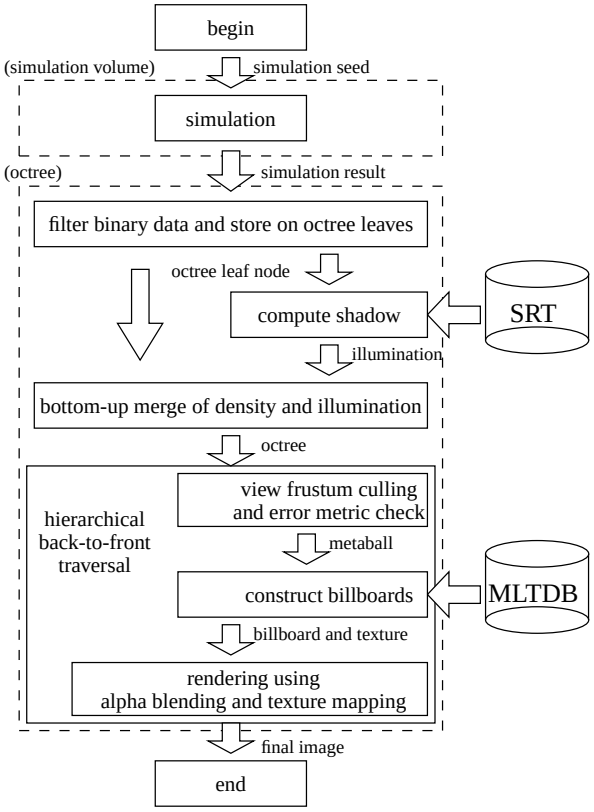


Fig. 4. Flowchart of run time process for clouds rendering.

to avoid the cloud growing or disappearing too much in a timestamp.

Based on the rules proposed by Dobashi et al. (2) and our modifications, the *cld* can be written as

$$cld\,(i,j,k,t_{i+1}) = \neg ext\,(i,j,k,t_i) \wedge (cld\,(i,j,k,t_i) \vee act\,(i,j,k,t_i))\,, \tag{5}$$

where *ext* is a new rule for cloud extinction, defined later in Eq. 12.

There are three types of vapor as follows:

$$hum\,(i,j,k,t_{i+1}) = IS\,(rnd < P_{cld2hum})\,, \tag{6}$$

$$hum\,(i,j,k,t_{i+1}) = hum\,(i,j,k,t_i) \wedge \neg act\,(i,j,k)\,, \tag{7}$$

$$\begin{aligned} hum\,(i,j,k,t_{i+1}) = {} & (hum\,(i,j,k,t_i) \wedge \neg act\,(i,j,k,t_i)) \\ & \vee (IS\,(rnd < P_{hum}\,(i,j,k,t_i)) \wedge f_{hum}\,(i,j,k))\,, \end{aligned} \tag{8}$$

where

$$\begin{aligned} f_{hum}\,(i,j,k) = {} & cld\,(i-1,j,k) \vee cld\,(i+1,j,k) \\ & \vee cld\,(i,j-1,k) \vee cld\,(i,j+1,k) \\ & \vee cld\,(i,j,k-1) \vee cld\,(i,j,k+1)\,. \end{aligned} \tag{9}$$

Eq. 6 is used when the clouds disappear, which have the possibility to transform to vapor depending on the cloud-to-vapor probability $P_{cld2hum}$. Eq. 7 is the same as the original rule in (2), which is used when the number of cloud voxels and vapor voxels reach the upper bound. Eq. 8 is a modified rule, which is used when the number of cloud voxels and vapor voxels does not reach the upper bound. Eq. 9 restricts the growth of vapor to be only in the boundary region of clouds.

The ability of transiting vapor to cloud can be rewritten as

$$\begin{aligned} act\left(i,j,k,t_{i+1}\right) = & \neg act\left(i,j,k,t_i\right) \wedge hum\left(i,j,k,t_i\right) \\ & \wedge \left(f_{act}\left(i,j,k\right) \vee IS\left(rnd < P_{act}\left(i,j,k,t_i\right)\right)\right), \end{aligned} \tag{10}$$

where

$$\begin{aligned} f_{act}\left(i,j,k\right) = & act\left(i-1,j,k\right) \vee act\left(i+1,j,k\right) \vee act\left(i,j-1,k\right) \\ & \vee act\left(i,j+1,k\right) \vee act\left(i,j,k-1\right) \vee act\left(i,j,k+1\right). \end{aligned} \tag{11}$$

Eq. 10 is the same as the original rule in (2). Based on Eq. 11, the affected region of *act* is smaller than that of the original rule in (2), and in consequence, balancing the abilities between the growth and extinction of clouds.

The extinction of clouds will work only when the number of cloud voxels greater than the lower bound, which is written as

$$\begin{aligned} ext\left(i,j,k,t_{i+1}\right) = & \left(\neg ext\left(i,j,k,t_i\right)\right) \wedge cld\left(i,j,k,t_i\right) \\ & \wedge IS\left(rnd < P_{ext}\left(i,j,k,t_i\right)\right) \wedge f_{ext}\left(i,j,k\right), \end{aligned} \tag{12}$$

where

$$\begin{aligned} f_{ext}\left(i,j,k\right) = & \left(\neg cld\left(i-1,j,k\right) \wedge \neg hum\left(i-1,j,k\right)\right) \\ & \vee \left(\neg cld\left(i+1,j,k\right) \wedge \neg hum\left(i+1,j,k\right)\right) \\ & \vee \left(\neg cld\left(i,j-1,k\right) \wedge \neg hum\left(i,j-1,k\right)\right) \\ & \vee \left(\neg cld\left(i,j+1,k\right) \wedge \neg hum\left(i,j+1,k\right)\right) \\ & \vee \left(\neg cld\left(i,j,k-1\right) \wedge \neg hum\left(i,j,k-1\right)\right) \\ & \vee \left(\neg cld\left(i,j,k+1\right) \wedge \neg hum\left(i,j,k+1\right)\right). \end{aligned} \tag{13}$$

Eq. 13 reveals that the clouds disappear only in the boundary region.

The rules for clouds advect by wind is the same as the rules in (2).

5.2 Rendering

A two-pass rendering method is used to render the cloud images. Sec. 5.2.1 describes the first pass rendering, which is used to calculate the illumination of all voxels. Sec. 5.2.2 introduces the second pass rendering, which traverses the volume in a back-to-front order, and renders the cloud images using textured billboards. In Sec. 5.2.3, a hierarchical texture caching scheme is proposed to speed up the rendering by reducing the number of billboards to be rendered.

5.2.1 First Pass Rendering

The first pass calculates the illumination of each voxel in the simulation volume using SRT. For each ray in SRT, we traverse the voxels associated with the ray in order. The illumination for each leaf voxel of octree is calculated using the following equation, which is rewritten from Eq. 1.

$$I_i = I_{i-1}(1.0 - D_i \sigma_a \tau), \tag{14}$$

where I_i is the illumination for the i_{th} voxel, and D_i is the voxel density.

5.2.2 Second Pass Rendering

The second pass renders the clouds in a back-to-front order by traversing the octree. The clouds are rendered using textured billboards associated with each leaf voxel. For each leaf node traversed, the texture for the voxel is obtained by first selecting the metaball texture from MLTDB using voxel density D_{voxel} and $\alpha_{L,E}$ as indices, and then multiplying it by I_i before rendering to texture. A back-to-front composition is performed as follows:

$$C_f(i+1) = C_b(i) + (1-D)C_f(i), \tag{15}$$

where $C_f(i+1)$ is the new color value to be filled into the frame-buffer, $C_f(i)$ is the original color value in the frame buffer, C_b is the voxel's textured billboard to be rendered, and D is the density. The view-frustum culling is also performed during the octree traversal to reduce the number of billboards to be rendered.

5.2.3 Speed-Up Scheme

Similar to the concept of level-of-detail modeling, a hierarchical texture caching is proposed to render the textured billboards in internal nodes, instead of leaf nodes whenever possible, aiming to reduce the number of billboards rendered. Instead of caching the rendered images as in (23; 26), we cache the metaball textures and billboards. Four tests of error metric are required to check if the cached information

can be re-used. They are distance test, angle test, voxel illumination test, and voxel density test, described as follows:

- **Distance test :** $\frac{R_l}{d} < \varepsilon_d$
  d is the current distance between voxel and eye point. R_l is the half length of voxel at level l. ε_d is the distance tolerance. This test is a variant of that in the work of Dobashi et al. (27) and controls the texture selection in different level depending on distance.
- **Angle test :** $\beta_{i+1} - \beta_i < \alpha_{tex,db}$
  β_i is the cached $\alpha_{L,E}$ value and β_{i+1} is the new $\alpha_{L,E}$ value. $\alpha_{tex,db}$ is the angle increment used for sampling $\alpha_{L,E}$ in MLTDB. The changes of $\alpha_{L,E}$ implies that the angle changes between the camera and the billboard, and affects the life cycle of both the cached texture and the cached billboard.
- **Voxel density test :** $\frac{D_{i+1}-D_i}{N_{voxel}} < \varepsilon_D$
  D_i and D_{i+1} are the cached voxel density and the new voxel density, respectively. N_{voxel} is the number of leaf nodes covered by the internal node. ε_D is the density tolerance. This test checks if the density variation in an internal node is under ε_D. If it is true, the same texture can be re-used.
- **Voxel illumination test :** $\frac{I_{i+1}-I_i}{N_{voxel}} < \varepsilon_I$
  I_i and I_{i+1} are the cached illumination and the new illumination, respectively. N_{voxel} is the number of leaf nodes covered by the internal node. ε_I is the illumination tolerance. This test checks if the illumination variation in an internal node is under ε_I. The cached metaball texture can be re-used if the test passes.

The distance and angle tests are used to decide if the cached billboard can be re-used after the viewing conditions change. The angle, voxel illumination, and voxel density tests are used to check if the cached metaball texture can be re-used after cloud density changes.

The hierarchical texture caching is embedded in the octree traversal. In the first pass rendering, after the illumination values of the leaf nodes are computed, the illumination I_{in} and the voxel density D_{voxel} of internal nodes are computed by a bottom-up process. The voxel density of an internal node is the maximum densities of its eight children, while the voxel illumination of an internal nodes is computed by averaging the illumination values of its eight children.

For each node in the octree traversal during the second pass rendering, if its cached information exists, the four tests for the node are examined. If all four tests succeed, the cached texture is re-used. Otherwise, the cached information is removed and each of its eight children is recursively traversed until reaching a node for which all tests succeed or until reaching the leaf node. A bottom-up process is then performed to update the cached information in internal nodes. The scattering illumination of the clouds in an internal node can be obtained by its metaball texture, selected from MLTDB using the voxel density and the current $\alpha_{L,E}$ value, multiplied by its voxel illumination.

6 Experimental Results

The programs are implemented in C/C++ and OpenGL and run on a dual AMD Opteron 2.2GHz CPUs machine with NVidia Quadro FX 4000 graphics card. Two threads are used, one for cloud simulation and the first pass rendering, the other for the second pass rendering and the hierarchical texture caching mechanism.

Table 2
Environment and parameter setting.

Parameter	Value
Wind speed	1
Wind Direction	West-to-East
Average P_{act}	0.01
Average P_{ext}	0.1
$P_{cld2hum}$	0.8
Cloud radius (macrostructure)	$6.0 \sim 10.0$
Lower bound of clouds and vapor	0.05
Size of octree leaf node	2.0
Sun angle	120°
σ_{s1}	0.2
σ_{s2}	2.0
Illumination attenuation ratio	0.07

We first describe the parameter settings that are common for our performance testing. In Table 2, P_{hum} is the probability of vapor, P_{act} is the vapor-to-cloud probability, P_{ext} is the probability of cloud extinction, and $P_{cld2hum}$ is the probability of cloud-to-vapor. Table 3 lists the other parameter settings for experiments $1 \sim 4$. The parameters include volume size, average P_{num}, upper bound on the number of cloud and vapor voxels, and eye position. Experiments 1, 2, and 3 have the same average P_{hum} and upper bound, but different voxel sizes and eye positions. The experiment 4 tests the performance of cloudy weather generated with larger average P_{hum} and upper bound. In all experiments, camera looks at the direction of north-west, and the clouds move from upper-left to lower right, which is from west to east.

Table 4 shows the results of the average timing performance and the average number of billboards rendered. The time required for simulation and the first-pass rendering are almost constant, and the time used for the second-pass rendering varies depending on the number of billboards used. Also, in our experiment, the time required for second-pass rendering is generally much more than the time for simulation and the first-pass rendering. Fig. 5 is the performance statistic of experiment

Table 3
Settings for experiments $1 \sim 4$.

Exp. no.	Volume	Avg. P_{hum}	Upper bound	Eye position
1	$64 \times 16 \times 64$	0.1	0.2	$(128.0, 34.0, 128.0)$
2	$64 \times 16 \times 64$	0.1	0.2	$(64.0, 34.0, 64.0)$
3	$128 \times 16 \times 128$	0.1	0.2	$(256.0, 34.0, 256.0)$
4	$64 \times 16 \times 64$	0.5	1.0	$(128.0, 34.0, 128.0)$

Table 4
Performance results of experiments $1 \sim 4$.

Exp. no.	Single-thread				Multi-thread	Avg. number of billboards
	Simulation time (ms)	First-pass time (ms)	Second-pass time (ms)	Avg. FPS	Avg. FPS	
1	13.685	11.176	27.412	15.017	30.606	12959
2	13.640	11.169	9.763	20.826	59.971	3438
3	53.737	45.901	101.241	4.6785	9.013	51109
4	14.959	11.722	43.588	12.028	22.365	22721

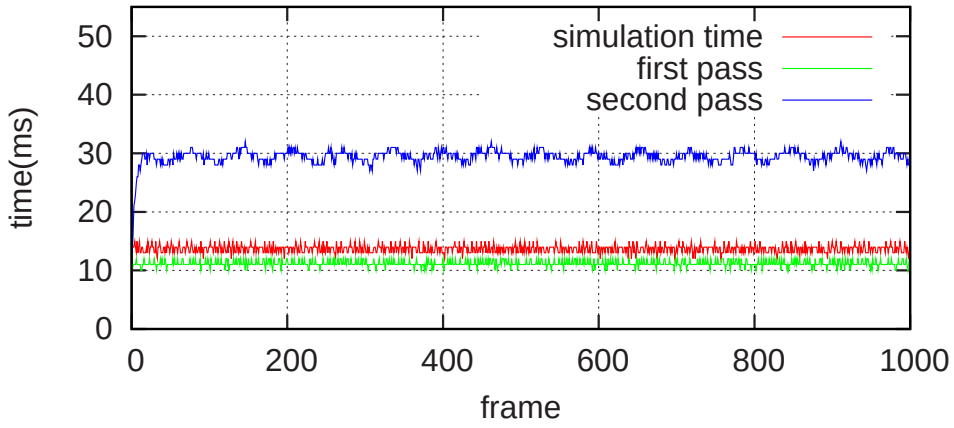
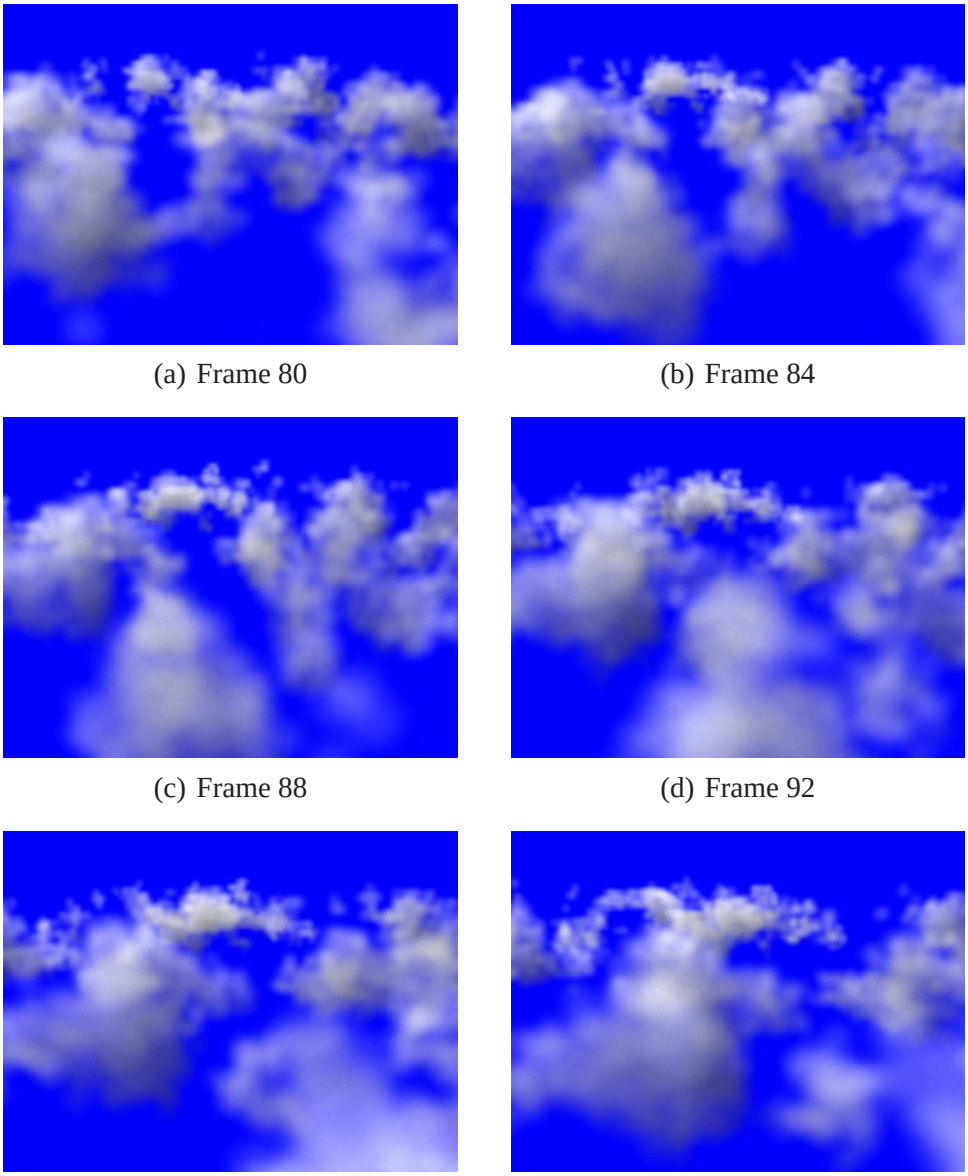


Fig. 5. Performance statistic of experiment 1.

1 in 1000 frames, and some of its cloud images are listed in Fig. 6.

Table 5 lists the average performances on a standard PC with a single Intel Pentium-4 3.0Ghz CPU and a NVidia Geforce 6800 graphics card. The simulation and first pass rendering are performed at an interval of 0.5 second and 1.0 second, and the second-pass rendering also blends the results of two consecutive simulation steps to get the in-between density and illumination values for rendering. The higher interval yields to slower advection in simulation, but better FPS performance.

Table 6 shows the parameter setting for hierarchical texture caching incorporated in experiments 1, 3, and 4. Table 7 depicts the results on FPS performance and image quality. Two image quality measurement methods are used. The root-mean-square



(a) Frame 80 (b) Frame 84

(c) Frame 88 (d) Frame 92

(e) Frame 96 (f) Frame 100

Fig. 6. Screenshot of cloud image.

is computed using

$$e_{rms} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x,y) - f(x,y) \right]^2 \right]^{\frac{1}{2}},$$

and the signal-to-noise ratio is computed as

$$SNR = 10 \times \log \left[\frac{\sum_{x=0}^{M-1} \hat{f}(x,y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x,y) - f(x,y) \right]^2} \right].$$

Fig. 7 illustrates the image quality statistic of experiment 1 over 500 frames, and

Table 5
Performance results of experiments $1 \sim 4$ using a standard PC with a single CPU.

Exp. no.	FPS in 0.5s interval	FPS in 1.0s interval
1	20.936	22.732
2	58.734	59.812
3	7.009	7.571
4	15.56	16.99

Table 6
Parameter setting for the hierarchical texture caching.

Exp. no.	ε_d	ε_D	ε_I
1	0.027	0.12	1.0
3	0.016	0.12	1.0
4	0.029	0.12	1.0

Table 7
FPS performance and image quality for experiments 1, 3, and 4 using hierarchical texture caching.

Exp. no.	FPS (multi-thread)		RMS	SNR
	without HTC	with HTC		
1	30.606	35.322	6.965	21.0783
3	9.013	18.021	8.509	20.623
4	22.365	31.574	10.003	22.990

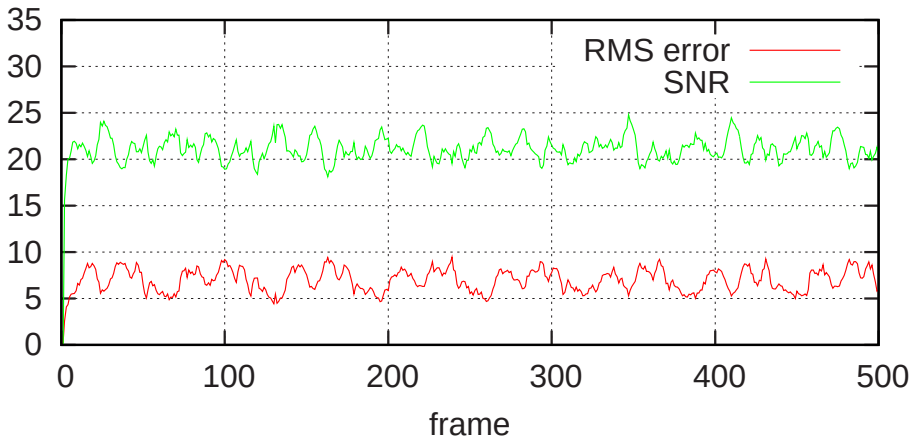
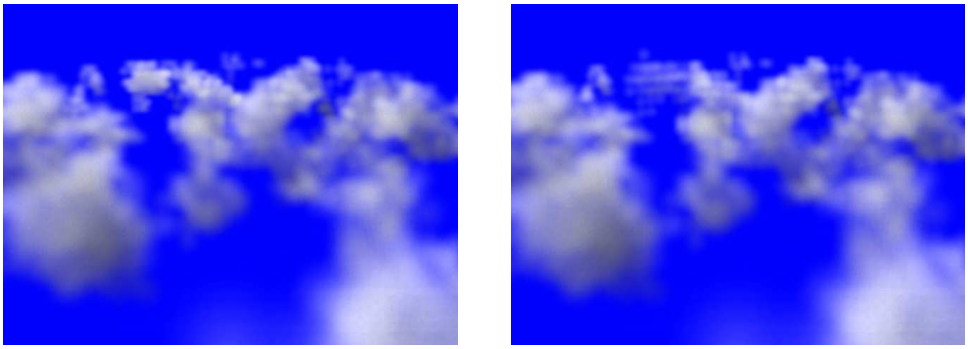


Fig. 7. Quality statistic of experiment 1 with hierarchical texture caching.

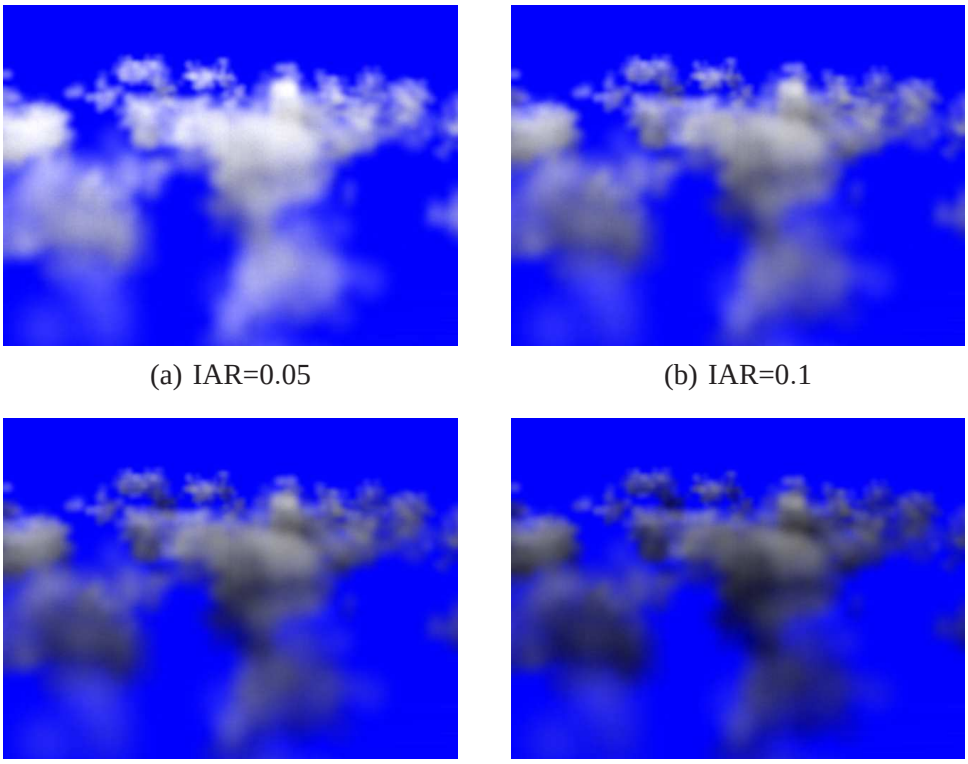
Fig. 8 shows some of cloud images rendered with and without hierarchical texture caching.



(a) Without HTC (b) With HTC

Fig. 8. Cloud images rendered with and without hierarchical texture caching.

Images in Fig. 9 are rendered using different illumination attenuation ratios (IAR). The higher IAR value, the faster the illumination attenuates, which results in more clear shadows. Fig. 10 are rendered in different pseudo scattering coefficients (σ_{s2}). Higher σ_{s2} value makes the clouds look more bright.

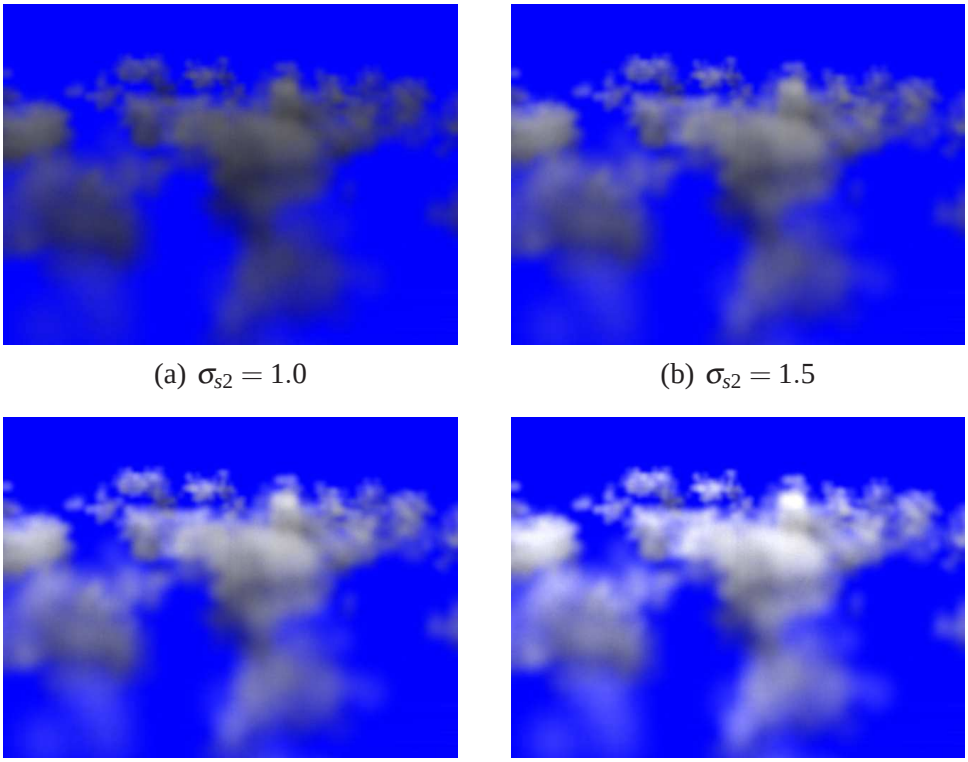


(a) IAR=0.05 (b) IAR=0.1

(c) IAR=0.15 (d) IAR=0.2

Fig. 9. Images rendered in different illumination attenuation ratios.

By controlling the simulation parameters and illumination parameters, we can simulate different weather conditions. Fig. 11 demonstrates some images of the different weathers.



(a) $\sigma_{s2} = 1.0$ (b) $\sigma_{s2} = 1.5$

(c) $\sigma_{s2} = 2.0$ (d) $\sigma_{s2} = 2.5$

Fig. 10. Images rendered in different pseudo scattering coefficient (σ_{s2}).

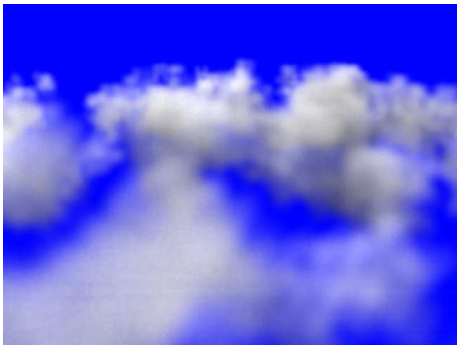
7 Conclusion

We have proposed an efficient rendering framework for dynamic clouds with features includes

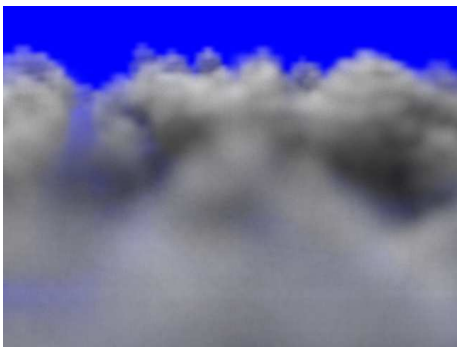
- A simplified lighting model which splits the lighting calculation into preprocessing and run-time stage in such a way that expensive computing steps are done in preprocessing.
- The SRT that records the spatial relationship of voxels along lighting direction. Based on SRT, illumination for each voxel can be determined very quickly.
- The MLTDB that stores the metaball texture for each density and viewing angle. Based on MLTDB, computing scattering illumination of a voxel involves a table look-up and a multiplication.
- The octree structure for efficient back-to-front traversal and view frustum culling.
- A hierarchical texture caching that aims to reduce the texture selection cost and the number of billboards.

The experiments have shown that, using the proposed techniques, nearly realistic cloud animation can be rendered in nearly real time on a standard PC, or in real time on a dual-CPUs PC.

(a) Sunny



(b) Cloudy



(c) Overcast

Fig. 11. Different weather conditions.

Our current implementation does not take into account the advantages of modern graphics hardware. The programmability of the modern graphics hardware should be explored to share the computation load of CPU and to speed up the simulation and rendering. The cellular automata we use for cloud simulation doesn't support many other kinds of clouds, such as stratus or cirrus. Other simulation methods, such as CML, may be applied to solve this problem.

References

[1] Y. Dobashi, T. Nishita, T. Okita, Animation of Clouds Using Cellular Automaton, in: Proceedings of Computer Graphics and Imaging'98, 1998, pp. 251–256.

[2] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, T. Nishita, A Simple, Efficient Method for Realistic Animation of Clouds, in: Proceedings of SIGGRAPH'00, 2000, pp. 19–28.

[3] R. Fedkiw, J. Stam, H. W. Jensen, Visual Simulation of Smoke, in: Proceedings of SIGGRAPH'01, 2001, pp. 15–22.

[4] N. Foster, D. Metaxas, Modeling the Motion of a Hot, Turbulent Gas, in: Proceedings of SIGGRAPH'97, 1997, pp. 181–188.

[5] R. Miyazaki, S. Yoshida, Y. Dobashi, T. Nishita, A Method for Modeling

Clouds Based on Atmospheric Fluid Dynamics, in: Proceedings of the 9th Pacific Conference, 2001, pp. 363–372.
[6] W. T. Reeves, Particle Systems–A Technique for Modeling a Class of Fuzzy Objects, in: Proceedings of SIGGRAPH'83, 1983, pp. 359–375.
[7] J. Stam, Stable Fluids, in: Proceedings of SIGGRAPH'99, 1999, pp. 121–128.
[8] J. Stam, Interacting with Smoke and Fire in Real Time, Communications of the ACM 43 (7) (2000) 76–83.
[9] J. Stam, A Simple Fluid Solver Based on the FFT, Journal of Graphics Tools 6 (2) (2001) 43–52.
[10] J. F. Blinn, Light Reflection Functions for Simulation of Clouds and Dusty Surfaces, in: Proceedings of SIGGRAPH'82, 1982, pp. 21–29.
[11] D. S. Ebert, Procedural Volumetric Cloud Modeling and Animation, SIGGRAPH'00 course notes 25 (5) (2000) 1–55.
[12] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, S. Worley, Texturing & Modeling, Morgan Kaufmann, 1998.
[13] G. Y. Gardner, Visual Simulation of Clouds, in: Proceedings of SIGGRAPH'85, 1985, pp. 297–304.
[14] H. W. Jensen, P. H. Christensen, Efficient Simulation of Light Transport in Scences with Participating Media using Photon Maps, in: Proceedings of SIGGRAPH'98, 1998, pp. 311–320.
[15] J. T. Kajiya, B. P. V. Herzen, Ray Tracing Volume Densities, in: Proceedings of SIGGRAPH'84, 1984, pp. 165–174.
[16] P. Elinas, W. Stuerzlinger, Real-Time Rendering of 3D Clouds, Journal of Graphics Tools 5 (4) (2001) 33–45.
[17] M. J. Harris, A. Lastra, Visual Simulation of Clouds, in: Proceedings of Eurographics'01, 2001, pp. 76–84.
[18] S. A. King, R. A. Crawfis, W. Reid, Fast Animation of Amorphous and Gaseous Phenomena, in: Volume Graphics'99, 1999, pp. 333–346.
[19] J. Stam, E. Fiume, A Multiple-Scale Stochastic Modeling Primitive, in: Proceedings of Graphics Interface'91, 1991, pp. 24–31.
[20] M. J. Harris, W. V. Baxter, T. Scheuermann, A. Lastra, Simulation of cloud dynamics on graphics hardware, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, Eurographics Association, 2003, pp. 92–101.
[21] Y. Dobashi, T. Nishita, H. Yamashita, T. Okita, Modeling of Clouds from Satellite Images Using Metaballs, in: Proceedings of the 6th Pacific Conference, 1998, pp. 53–60.
[22] K. Perlin, An Image Synthesizer, in: Proceedings of SIGGRAPH'85, 1985, pp. 287–296.
[23] G. Schaufler, Dynamically Generated Imposters, GI Workshop "Modeling – Virtual Worlds – Distributed Graphics" (1995) 192–135.
[24] G. Wyvill, C. McPheeters, B. Wyvill, Data Structure for Soft Objects, The Visual Computer 2 (4) (1986) 227–234.
[25] K. Nagel, E. Raschke, Self-Organizing Criticality in Cloud Formation?, Physica A (1992) 519–531.

[26] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, J. Snyder, Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments, in: Proceedings of SIGGRAPH'96, 1996, pp. 75–82.
[27] Y. Dobashi, T. Yamamoto, T. Nishita, Efficient Rendering of Lightning Taking into Account Scattering Effects due to Clouds and Atmospheric Particles, in: Proceedings of the 9th Pacific Conference, 2001, pp. 390–399.
[28] D. S. Ebert, R. E. Parent, Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-buffer Techniques, in: Proceedings of SIGGRAPH'90, 1990, pp. 357–366.
[29] T. Nishita, Y. Dobashi, Modeling and Rendering Methods of Clouds, in: Proceedings of the 7th Pacific Conference, 1999, pp. 218–219.
[30] T. Holtkämper, Real-Time Gaseous Pheonema - A Phenomenological Approach to Interactive Smoke and Steam, in: Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa, 2003.
[31] A. Fujimoto, T. Tanaka, K. Iwata, ARTS: Accelerated Ray Tracing System, IEEE Computer Graphics & Applications 6 (4) (1986) 16–26.
[32] L. Westover, Footprint Evaluation for Volume Rendering, in: Proceedings of SIGGRAPH'90, 1990, pp. 367–376.
[33] S. J. Gortler, R. Grzeszczuk, R. Szeliski, M. F. Cohen, The Lumigraph, in: Proceedings of SIGGRAPH'96, 1996, pp. 43–54.
[34] J. Shade, S. Gortler, L. W. He, R. Szeliski, Layered Depth Images, in: Proceedings of SIGGRAPH'98, 1998, pp. 231–242.
[35] P. Wesseling, An Introduction to Multigrid Methods, John Wiley & Sons, 1992.
[36] E. Kmett, Harmless Algorithms – Scene Traversal Algorithms, http://www.flipcode.com/harmless/issue02.htm (1999).
[37] T. Möller, E. Haines, Real-Time Rendering, A K Peters Ltd, 1999, Ch. 10.7 Plan/Box Intersection Detection, pp. 310–313.
[38] N. Miller, Billboarding, http://nate.scuzzy.net/docs/billboard/ (2000).
[39] T. Nishita, E. Nakamae, A Method for Displaying Metaballs by Using Bezier Clipping, in: Proceedings of Eurographics'94, 1994, pp. 271–280.