A Simple, Efficient Method for Realistic Animation of Clouds

Yoshinori Dobashi[*] Kazufumi Kaneda[**] Hideo Yamashita[**] Tsuyoshi Okita[*] Tomoyuki Nishita[***]

[*]Hiroshima City University [**]Hiroshima University [***]University of Tokyo
[*]{doba, okita}@im.hiroshima-cu.ac.jp [**]{kin, yama}@eml.hiroshima-u.ac.jp [***]nis@is.s.u-tokyo.ac.jp

Abstract

This paper proposes a simple and computationally inexpensive method for animation of clouds. The cloud evolution is simulated using cellular automaton that simplifies the dynamics of cloud formation. The dynamics are expressed by several simple transition rules and their complex motion can be simulated with a small amount of computation. Realistic images are then created using one of the standard graphics APIs, OpenGL. This makes it possible to utilize graphics hardware, resulting in fast image generation. The proposed method can realize the realistic motion of clouds, shadows cast on the ground, and shafts of light through clouds.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture; I.3.6 [Computer Graphics:] Methodology and Techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.6.3 [Simulation and Modeling]: Applications;

Additional Keywords: Animation, Atmospheric Effects, Rendering, Graphics Hardware, Volume Rendering.

1. INTRODUCTION

Clouds play an important role when making images for flight simulators or outdoor scenes. Their color and shapes change depending on the position of the sun and the observer. This means that the density distribution of clouds should be defined in three-dimensional space to create realistic images. Therefore, a lot of methods have been developed to display clouds [33, 10, 5, 14, 19, 30, 25]. Using these methods, extremely realistic images can be generated. Their main purpose is, however, to create images of static clouds. Fascinating animations of clouds with changing their shapes and color are often used, however, in movies, commercial films and so on. They are often created by filming them in advance and replaying the film quickly. Since generating such realistic animations by computer graphics is useful, a lot of methods have been developed [13, 29, 6, 31, 7, 22, 9, 15, 8, 32].

This paper proposes a new method for realistic animation of clouds. Our aim is to develop a simple method that can create realistic animation as quickly as possible, preferably in real-time. We propose an efficient simulation method and a hardware-accelerated rendering method. In particular, the method is suitable for animation of cumulus-like clouds in landscape scale. Our method has the following features.

- Our simulation method creates realistic cloud motion with a small amount of computation. It uses cellular automaton that can simulate the motion just by simple Boolean operations.
- Our rendering method realizes a fast computation of photo-realistic images. It can quickly calculate shadows and shafts of light through clouds, as well as cloud color, by making the most of graphics hardware.

A straightforward approach to creating realistic cloud motion is to simulate the physical phenomena. That, however, is impractical since it is computationally expensive. Therefore, a simple and efficient method is required that maintains the visually convincing result. As one such methods, Nagel et al. extremely simplified the cloud dynamics using cellular automaton [21]. The method can simulate cloud formation by simple transition rules. Unfortunately, the method is not sufficient for our purpose since their aim is not to create realistic animation. So, we extend their work and propose a new method to realize realistic cloud evolution.

Furthermore, we propose a method for generating realistic images including cloud shadows and shafts of light using the standard graphics API, OpenGL. This results in fast image generation since we can make use of graphics hardware. Our rendering method can display the following three effects: 1) cloud color taking into account the single scattering of light, 2) shadows of clouds cast on the ground, 3) shafts of light through clouds. Most of the previous methods render these effects by using ray-tracing, one of the most time-consuming methods. To overcome this problem, we propose a hardware-accelerated rendering method based on OpenGL. Using the proposed method, a photo-realistic image can be generated within one minute on a standard PC.

Our method is not sufficient if the user needs physically exact cloud motion, since it is one of the numerical models that simplifies the physical phenomena. Our method is suitable for users who want a simple, easy-to-use, and computationally inexpensive method that can create visually convincing results. Furthermore, the proposed method makes as much use of graphics hardware as possible. Since graphics hardware is becoming faster and faster, we believe our method is one of the promising techniques that will realize real-time animation in the near future.

*3-4-1, Ozukahigashi, Asaminami-ku, Hiroshima, 731-3194 Japan
(Dobashi's current address: Hokkaido University, Faculty of Engineering, Kita 13, Nishi 8, Kita-ku, Sapporo 060-8628 Japan)
**1-4-1, Kagamiyama, Higashi-hiroshima, 739-8527 Japan
***7-3-1, Hongo, Bunkyo-ku, Tokyo, 113-0033 Japan

SIGGRAPH 2000, New Orleans, LA USA
© ACM 2000 1-58113-208-5/00/07 ...$5.00

2. PREVIOUS WORK

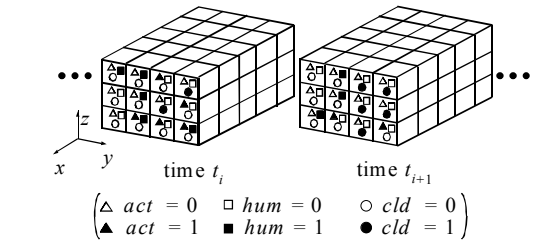
In this section, previous works are briefly reviewed. Methods related to the simulation and methods to rendering are separately reviewed.

SIMULATION: In computer graphics, there are two categories to simulate the gaseous motion like clouds. One is to simulate the physical process of fluid dynamics [13, 29, 31, 9, 32]. The other is a heuristic approach [10, 6, 7, 22, 15, 8]. Most of the methods in the former category need a large amount of computation time. Stam, however, developed a fast simulation method by simplifying fluid dynamics [32]. He demonstrated a real-time animation of smoke on a high-end workstation. However, since our purpose is to simulate clouds covering a large area in the sky using a standard PC, it is sill time-consuming. Furthermore, the phase transition effects from vapor to water should be incorporated to simulate cloud formation. To our knowledge, the method developed by Kajiya et al. is the only one to include the phenomena [13]. It is, however, very complex and time-consuming. Controlling cloud shapes is also difficult by using the methods in this category.

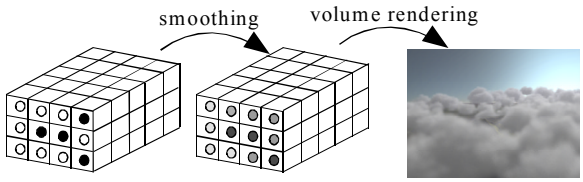
On the other hand, the latter approaches, such as procedural modeling, are computationally inexpensive and much easier to implement. The disadvantage in these methods, however, is that the user has to search parameters by trial and error to create realistic animation. Creating realistic-looking motion using them does not seem to be an easy task.

Our method lies in an intermediate position between these two categories. The method reflects the physical formation process of clouds in part, and it is computationally more efficient and easier to implement than are the previous physical simulation methods. Our method can create more realistic cloudy scenes than the physical based methods do. Controlling the shapes and their motion to create realistic animation is an easy task using our method. Dobashi et al. tried to develop such a method using cellular automaton [3]. Unnatural animation is created by that, however, since the formation and extinction of clouds are frequently repeated. The method in this paper is an extension to their method.

RENDERING: One of the simplest ways to display clouds is to use mapping techniques, such as the method developed by Gardner [10]. In order to display photo-realistic images, however, it is desirable to use the physical model, taking into account scattering/absorption due to particles. Many such methods have therefore been developed [13, 5, 14, 28, 29, 19, 31, 4, 32]. Some of them take into account multiple scattering of light [13, 19, 31, 25]. Additionally, Nishita et al. take into account the effect of skylight on the cloud color [25]. Multiple scattering and skylight are important for realistic image synthesis but is time-consuming. Our method approximates them as a constant ambient term. One of the major approaches to rendering the volume density similarly to clouds is to use 3D textures. Stam used 3D hardware texture mapping to display gases [32]. With the help of the high-end graphical workstation, the method can generate realistic images in real-time by combining 3D textures and advecting cloud textures developed by Max et al. [18]. Unfortunately, his method is not sufficient for our purpose since the method does not include the atmospheric effects such as shafts of light, one of the essential factors in generating realistic images of outdoor scenes. Our method can handle these effects. Although using 3D textures is simpler and efficient, the 3D texture mapping hardware is still expensive and not universally available as are 2D textures. Since one of our objectives is to create realistic animation on a standard PC, we propose a method using 2D textures. The idea of the method is of course applicable to 3D textures.



(a) Simulation process.



(b) Rendering process.

Figure 1: Overview of our method.

There are also a lot of methods for generating shafts of light [23, 11, 12, 16, 17, 27, 24]; most of them use the ray-tracing algorithm and require dozens of minutes. Although Nishita and Nakamae developed a faster method using a scanline accumulation buffer [24], it still requires several minutes. We propose a much faster method using hardware color blending and texture mapping functions. To our knowledge, no other methods are available making use of graphics hardware to render shafts of light.

3. BASIC IDEA

Fig. 1 shows an overview of our method. Our method consists of two processes, simulation and rendering. As shown in Fig. 1(a), the simulation space is divided into voxels. The voxels correspond to cells used in the cellular automaton. At each cell, three logical variables, vapor/humidity (*hum*), clouds (*cld*), and phase transition (or activation) factors (*act*) are assigned. The state of each variable is either 0 or 1. Cloud evolution is simulated by applying simple transition rules at each time step. The transition rules represent formation, extinction, and advection by winds. Since the state is either 0 or 1, the rules can be expressed by Boolean operations. Therefore, each variable can be stored in one bit to save the memory cost and the simulation process is accelerated by using bit field manipulation functions.

Images are generated in the rendering process by making use of the simulation results (Fig. 1(b)). As described above, what we can obtain from the simulation is no more than *there are clouds* (*cld* = 1) or, *there are not-clouds* (*cld* = 0) at each voxel. Therefore, a density at each point is calculated by smoothing the binary distribution as shown in Fig. 1(b). The clouds are then rendered using volume rendering techniques. The rendering process consists of two steps. The first step calculates the intensity of light reaching the center of each voxel. Cloud shadows are also calculated in this step. The shadows are obtained as a texture. Then, in the second step, images are generated. Clouds are rendered by using a splatting method [1]. To render shafts of light, we consider multi spherical shells with their center at the viewpoint (see Fig. 7). The shells are then drawn from back to front using the hardware alpha-blending function. Shafts of light

are rendered by mapping the shadow texture on the shells (see section 5.2).

4. SIMULATION METHOD

We extend the following four points to Nagel's method [21]:

- Extinction of clouds
- Wind effects
- Speeding up of the simulation
- Controlling cloud motion

Details of the above extensions are explained in sections 4.2 through 4.5 after the brief description of Nagel's method in the next section.

4.1 Growth Simulation

In this section, Nagel's method to simulate the cloud formation process is described briefly. The physical processes of cloud formation are outlined as follows. Clouds are formed as a bubble of air is heated by underlying terrain heat, causing the bubble to become less dense, and to rise into regions of lower pressure in which the bubble expands. Expansion cools the bubble, increasing the relative humidity inside the bubble. The phenomenon called phase transition then occurs, that is, water vapor in the bubble becomes water droplets, or clouds. Nagel et al. used a cellular automaton [35] to simulate these processes in the following way.

For simplicity, the simulation space is aligned parallel to *xyz* axes and the number of cells is assumed to be $n_x \times n_y \times n_z$. As mentioned before, three logical variables, *hum*, *act*, and *cld*, are assigned at each cell (see Fig. 1(a)). Each represents vapor, phase transition factor, and clouds. The state of each variable is either 0 or 1. *hum*=1 means there is enough vapor to form clouds, *act*=1 means the phase transition from vapor to water (clouds) is ready to occur, and *cld*=1 means there are clouds. In the following, $A \wedge B$ and $A \vee B$ indicate conjunction and disjunctinon between A and B, respectively, and $\neg A$ indicates negation of A. Their transition rules are given as follows.

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

$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \vee act(i,j,k,t_i), \quad (2)$$

$$act(i,j,k,t_{i+1}) = \neg act(i,j,k,t_i) \wedge hum(i,j,k,t_i) \wedge f_{act}(i,j,k), \quad (3)$$

where $f_{act}(i,j,k)$ is a Boolean function and its value is calculated by the status of *act* around the cell. The following function is used by taking into account the fact that clouds grow upward and horizontally.

$$\begin{aligned} f_{act}(i,j,k) = {} & act(i+1,j,k,t_i) \vee act(i,j+1,k,t_i) \\ & \vee act(i,j,k+1,t_i) \vee act(i-1,j,k,t_i) \vee act(i,j-1,k,t_i) \\ & \vee act(i,j,k-1,t_i) \vee act(i-2,j,k,t_i) \vee act(i+2,j,k,t_i) \\ & \vee act(i,j-2,k,t_i) \vee act(i,j+2,k,t_i) \vee act(i,j,k-2,t_i). \end{aligned} \quad (4)$$

Of course, there are variations of the above function. We have tried some of them and we couldn't see significant differences of the simulation. So, we use the function in the original paper. The rules are summarized in Fig. 2. As shown in the top column of Fig. 2, *act* becomes 1 if *hum* is 1 and the state of *act* of one of the shaded cells around the center cell (*i*, *j*, *k*) is 1. Then, *hum* becomes 0 as shown in the middle. Finally, as shown in the bottom of Fig. 2, *cld* becomes 1. As a boundary condition, their states are assumed to be 0 outside the simulation space. Beginning from initial random status (all 3 status are set randomly), cloud growth is simulated by updating the state of each variable using Eqs. 1 through 4. As for the initialization, *hum* and *act* are determined randomly and *cld* is set to zero. For more details, please refer to [21].

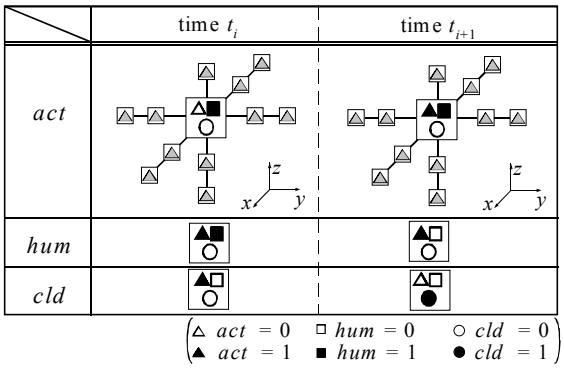


Figure2: Basic transition rules.

4.2 Cloud Extinction

One of the disadvantages of Nagel's method is that cloud extinction never occurs since *cld*, after it has become 1, remains 1 forever. Dobashi et al. introduced a new state variable to solve this problem [3]. In that method, however, formation and extinction are repeated frequently, resulting in unnatural animation. In the real world, cloud extinction is caused by gradual transition of water droplets to vapor. Our method simulates the cloud extinction as follows. First, the animator specifies cloud extinction probability, p_{ext}. Next, at each cell whose *cld* is 1, a random number, *rnd* ($0 \le rnd \le 1$), is generated and *cld* is changed to 0 if $rnd < p_{ext}$. By changing the probability at each cell at different times, the animator can specify regions where cloud extinction occurs frequently. Although this realizes the cloud extinction, there remains another problem. Clouds are never generated after the extinction at the cell. To solve this, vapor (*hum*) and phase transition factors (*act*) are supplied at specified time intervals. Similar to extinction, vapor probability, p_{hum}, and phase transition probability, p_{act}, are used to set them randomly. That is, *hum* is changed to 1 if $rnd < p_{hum}$ and *act* is changed to 1 if $rnd < p_{act}$. Cloud motion can be controlled by controlling the probabilities, p_{hum}, p_{act}, and p_{ext} at each cell at each time step. The methods described in this section are summarized by the following three transition rules.

$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \wedge \mathrm{IS}(rnd > p_{ext}(i,j,k,t_i)), \quad (5)$$

$$hum(i,j,k,t_{i+1}) = hum(i,j,k,t_i) \vee \mathrm{IS}(rnd < p_{hum}(i,j,k,t_i)), \quad (6)$$

$$act(i,j,k,t_{i+1}) = act(i,j,k,t_i) \vee \mathrm{IS}(rnd < p_{act}(i,j,k,t_i)), \quad (7)$$

where *rnd* is a uniform random number, IS(*e*) is a Boolean function that returns 1 if the expression *e* is true, otherwise returns 0.

4.3 Advection by Wind

We can observe clouds moving in one direction, blown by winds. New transition rules are introduced to include the wind effect. The idea is simply to shift all the variables toward the wind direction. We assume, for simplicity, the wind blows toward the direction of *x*-axis. Other cases can be handled by rotating the simulation space according to the wind direction. Furthermore, it is well known that the wind velocity is different depending on the height from the ground. The wind velocity, $v(z_k)$, is therefore specified as a function of *z*-coordinate of each cell (*i*,*j*,*k*). To implement the wind effect in the context of the cellular automaton, the function,

$v(z_k)$, is assumed to return integer values. The transition rules are as follows.

$$hum(i,j,k,t_{i+1}) = \begin{cases} hum(i - v(z_k), j, k, t_i), & i - v(z_k) > 0 \\ 0, & \text{otherwise} \end{cases}, \quad (8)$$

$$cld(i,j,k,t_{i+1}) = \begin{cases} cld(i - v(z_k), j, k, t_i), & i - v(z_k) > 0 \\ 0, & \text{otherwise} \end{cases}, \quad (9)$$

$$act(i,j,k,t_{i+1}) = \begin{cases} act(i - v(z_k), j, k, t_i), & i - v(z_k) > 0 \\ 0, & \text{otherwise} \end{cases}, \quad (10)$$

In this paper, the velocity function $v(z_k)$ is specified as a piecewise linear function.

4.4 Fast Simulation Using Bit Field Manipulation Functions

Each variable can be stored in one bit since its state is either 0 or 1. This means that simulations with large numbers of cells can be executed in a small amount of memory. The computation time is also reduced because of the following reasons. Let us assume all the variables are stored in an array of unsigned integers. Let *m* be the bit length of the unsigned integer variable. By making use of bit field manipulation functions of higher level language, such as C language, transitions of *m* cells can be computed at the same time. This realizes fast simulation. Most difficulties in implementing this idea lie in transition rules concerning cloud extinction, expressed by Eqs. 5 through 7, since random numbers have to be generated at each cell, i.e., each bit field. This may result in increasing the computation time. We used a look-up table that stores random bit sequences to save the computational cost. See Appendix A for more details.

4.5 Controlling Cloud Motion Using Ellipsoids

As mentioned in section 4.2, the animator can design the cloud motion by controlling vapor probability, phase transition probability, and cloud extinction probability. Ellipsoids are used to do this in this paper. When wet air parcels move upward and reach the height of the dew point, clouds are gradually formed. Ellipsoids are used to simulate the air parcels. The vapor probability and phase transition probability are assumed to be higher at their centers than at their edges. Inversely, the cloud extinction probability is assumed to be lower at the center since the extinction hardly ever occurs at the center of the air parcel. Ellipsoids also move in the direction of the wind. By controlling ellipsoid parameters, such as sizes and positions, different kinds of clouds can be simulated. The animator specifies the regions for ellipsoids to be generated. In our experiment, even the ellipsoids generated using uniform random numbers result in a realistic animation as shown in section 6.

5. RENDERING METHOD

Methods for generating realistic images are proposed in this section. First, the density distribution of clouds is calculated by making use of the results of the simulation. Images are then rendered using OpenGL. Details of the methods are described in the following sections.

5.1 Continuous Density Distribution Calculation

The density distribution of clouds in the real world is continuous from 0 to 1. The distribution obtained from the simulation, however, has only two values, that is, 0 or 1. Therefore, the proposed method calculates continuous distribution by smoothing the binary distribution, or two-valued distribution. First, the density, $q(i,j,k,t_i)$, is calculated at each cell (i,j,k) at each time step t_i using the following equation.

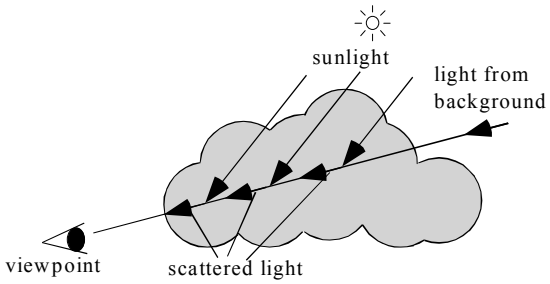


Figure 3: Calculation of cloud color.

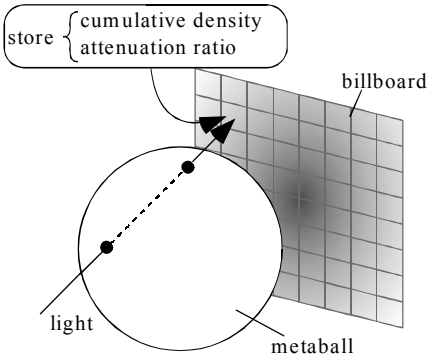


Figure 4: Billboard and its texture.

$$q(i,j,k,t_i) = \frac{1}{(2t_0+1)(2k_0+1)(2j_0+1)(2i_0+1)} \sum_{t'=-t_0}^{t_0} \sum_{k'=-k_0}^{k_0} \sum_{j'=-j_0}^{j_0} \sum_{i'=-i_0}^{i_0} w(i',j',k',t') cld(i+i', j+j', k+k', t_i+t'), \quad (11)$$

where w is a weighting function and i_0, j_0, k_0, t_0 are sizes for the smoothing. As expressed by Eq. 11, we include time as well as space for the smoothing since the distribution is discrete in space and time. The density at an arbitrary point, $\mathbf{x}$, is then obtained as a weighted sum of a simple basis function, f. Gaussians are often used for the basis function [29, 31, 34]. In this paper, however, we use a field function of metaballs proposed by Wyvill et al [36]. The reason for this is as follows. A metaball has a parameter, an effective radius, which represents its size. This means that it is much easier to specify the domain of influence than Gaussians that have an infinite domain. Furthermore, the shape of Wyvill's field function is very similar to the Gaussians [36]. As a result, the density at point $\mathbf{x}$ is given by the following equation.

$$\rho(\mathbf{x}, t_i) = \sum_{i,j,k \in \Omega(\mathbf{x},R)}^{N} q(i,j,k,t_i) f(|\mathbf{x} - \mathbf{x}_{i,j,k}|), \quad (12)$$

where R is the effective radius, $\Omega(\mathbf{x},R)$ is a set of cells those centers are within the distance R from the point $\mathbf{x}$, N is the number of elements of $\Omega(\mathbf{x},R)$, and $\mathbf{x}_{i,j,k}$ is the coordinate corresponding to the center of the cell (i,j,k). For the field function, f, see Appendix B. As shown in Eq. 12, the continuous density distribution is expressed by a set of metaballs. The user specifies the effective

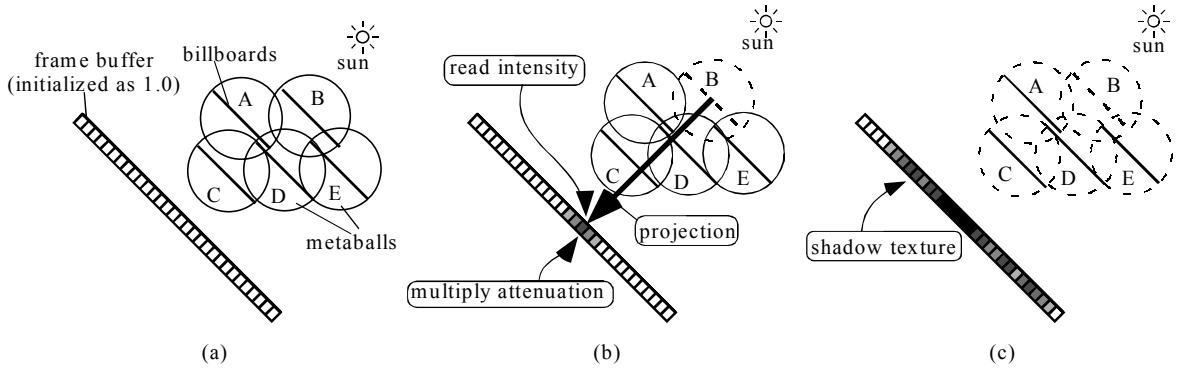


Figure 5: Algorithm for calculating the intensity of light reaching the center of metaballs. (a) Billboards are placed at the centers of metaballs and sorted based on their distances from the sun. The frame buffer is initialized as 1.0. (b) Billboards are projected onto the image plane. The colors in the frame buffer are multiplied by attenuation stored in billboard textures. (c) Shadow texture is obtained in the frame buffer. Each element stores the attenuation of light passing through clouds.

radius, R**.** The time step does not correspond to animation frames. Therefore, the density $\rho(\mathbf{x},t)$ at time t that corresponds to each animation frame is calculated by the linear interpolation of densities, $\rho(\mathbf{x},t_k)$ and $\rho(\mathbf{x},t_{k+1})$, where $t_k < t < t_{k+1}$.

5.2 Hardware-accelerated Rendering Using OpenGL

This section describes methods for displaying clouds as well as shafts of light using graphics hardware. Note that the algorithm is illustrated by a pseudo-code in appendix C.

5.2.1 Rendering Clouds

Rendering of clouds is based on the splatting algorithm using billboards. Details of the splatting method are well described in [34, 1, 20]. Therefore, let us omit the details. The basic idea for applying it to cloud display is described here.

Fig. 3 shows the idea of calculating the color of clouds taking into account the single scattering of light. First, the sum of the scattered light reaching from the sun on the viewing ray is calculated. The attenuated light reaching from behind the clouds is also calculated. The light reaching the viewpoint is the sum of those two. Therefore, the color of a voxel depends on the scattered color of the sun, the transmitted color of the sky, and the attenuation due to cloud particles. Calculation of cloud color using splatting is as follows. First, as shown in Fig. 4, textures for billboards are precalculated. Each element of the texture stores the attenuation ratio and cumulative density of the light passing through the metaball (see Fig. 4). Since the attenuation is not proportional to it, the texture has to be prepared for all meatballs when their center densities are different. However, this requires a large amount of memory. So, the density is discretized into n_q levels and n_q textures are prepared. In this paper, n_q is 64. The texture corresponding to the nearest density of each metaball is mapped onto the corresponding billboard. An image is calculated in two steps using the texture-mapped billboards. In the first step, the intensity of the light is calculated reaching from the sun at each metaball. The shadows of the clouds are also calculated in this step. In the second step, the image viewed from the viewpoint is generated. The two steps are as follows.

Fig. 5 shows the idea of the first step. The basic idea is to calculate an image viewed from the sun direction to obtain the intensity of light reaching each metaball. First, the viewpoint is placed at the sun position and the parallel projection is assumed. The frame buffer is initialized as 1.0. Then the billboards are placed at the center of each metaball with their normals oriented to the sun direction as shown in Fig. 5(a). Next, attenuation ratio between the center of each metaball and the sun is calculated. For example, the attenuation ratio between the metaball C and the sun is obtained by multiplying the attenuation ratio of metaballs A, B, and D (see Fig. 5(a)). To do this for all metaballs, the billboards are sorted in ascending order using the distance from the sun (the order is B-E-A-D-C in Fig. 5). Then, beginning from metaball B, they are projected onto the image plane. The values in the frame buffer are multiplied by their attenuation ratios that are stored in the billboard texture (Fig. 5(b)). This can be easily done by using blending functions of OpenGL. Then the pixel value corresponding to the center of the metaball is read from the frame buffer. The value obtained is the attenuation ratio between the sun and the metaball. The color of the metaball is obtained by multiplying the pixel value by the sunlight color. These processes are repeated for all metaballs. After all the metaballs are processed, the image in the frame buffer stores the attenuation ratio of the sunlight passing through the clouds (Fig. 5(c)). The image is

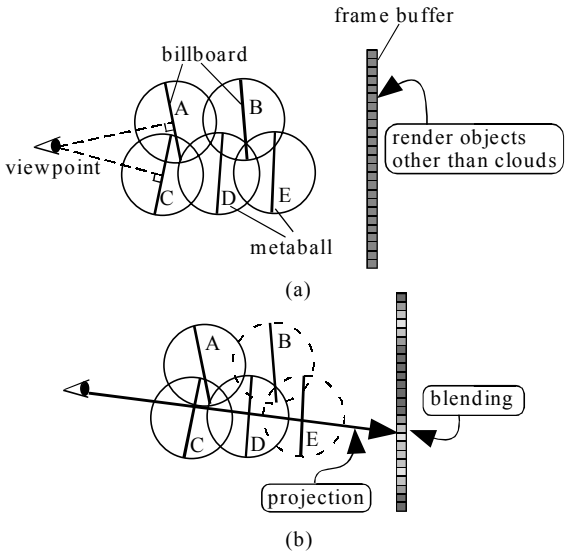


Figure 6: Algorithm for generating images. (a) Billboards are oriented to the viewpoint and sorted based on their distances from the viewpoint. (b) Billboards are projected onto the image plane. The colors in the frame buffer are attenuated and blended with the colors in the billborad textures.

stored as a light map texture [1] to cast shadows on the ground.

In the second step, the image is generated by using the color of the metaball obtained in the first step. First, all the objects except clouds are rendered. Next, as shown in Fig. 6(a), the billboards are faced perpendicularly to the viewpoint and sorted in descending order based on distances from the viewpoint (the order is E-B-D-A-C). Then they are projected onto the image plane in back-to-front order (Fig. 6(b)). The color in the frame buffer is blended with that of the billboard texture. The blending process is the same as the one used in the splatting method (see [1]). That is, the colors in the frame buffer are multiplied by the attenuation ratio of the billboard texture and then the colors in the texture are added. The process is repeated for all metaballs.

5.2.2 Rendering Shafts of Light

Fig. 7 shows the idea of calculating the shafts of light. Shafts of light are caused by particles in the atmosphere. The sunlight passing through gaps in clouds is scattered by the particles at P in Fig. 7. The scattered light, I_s, reaching the viewpoint is recognized as shafts of light. The scattering/absorption due to the atmospheric particles must therefore be taken into account. The intensity of light reaching the viewpoint is obtained by the following equation.

$$I = I_c \beta(T) + \int_0^T \gamma(s) I_s(s) \beta(s) ds \ , \tag{13}$$

where I_c is the cloud color, $\beta(s)$ is the attenuation ratio from the viewpoint to P due to atmospheric particles, $\gamma(s)$ is the attenuation ratio due to cloud particles from the sun to P, and $I_s(s)$ is the intensity of the light scattered at P due to atmospheric particles. The first term in the right hand side of Eq. 13 indicates the attenuation ratio of the intensity of light from clouds. The second term is related directly to the shafts of light. Preetham et al. take into account the scattering of sky light as well as the sunlight to render the aerial perspective [26]. In this paper, however, we ignore the scattering of sky light since it has little effect on shafts of light. We also assume that the density of the atmospheric particles decreases exponentially to the height from the ground. Under these assumptions, the attenuation ratio, $\beta(s)$, and the scattered light, $I_s(s)$, can be calculated analytically based on the positions of the viewpoint and P [14]. Furthermore, we assume the shafts of light are only visible under the cloud bottom, z_c. That is, the attenuation due to clouds, $\gamma(s)$, is 1.0 if $z_p > z_c$, where z_p is the z coordinate of P. In this case, $\gamma(s)$ is the attenuation ratio of the sunlight passing through the clouds. This means it has been stored in the shadow texture obtained in the first step described in the previous section. The shafts of light are rendered by calculating Eq. 13 as follows.

Eq. 13 is discretized as the following equation.

$$I = I_c \beta(T) + \sum_{k=0}^{n_s} \gamma(k\Delta s) I_s(k\Delta s) \beta(k\Delta s) \Delta s \ , \tag{14}$$

where n_s and Δs are the number of samples and the sampling interval for the integral in Eq. 13, respectively. The attenuation, $\beta(T)$, in the first term is calculated analytically by the positions of the viewpoint and each metaball. The color of each metaball is then attenuated by multiplying it. To calculate the second term, spherical shells are considered as shown in Fig. 7. Their centers are placed at viewpoint and their radii are determined so that the intervals of shells coincide to Δs. The shells are approximated by a set of polygons to render them using OpenGL. Polygons outside the viewing pyramid are discarded. Next, the intensity of the light scattered at each vertex and the attenuation ratio of the path between the viewpoint and the vertex are calculated. Then

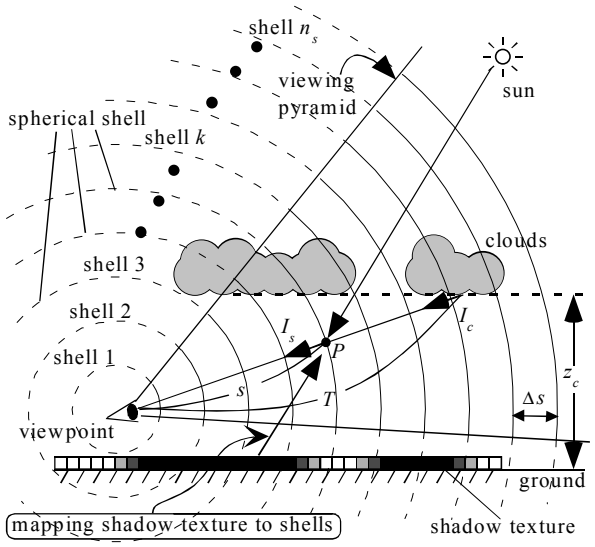


Figure 7: Rendering shafts of light.

$I_s(k\Delta s)\beta(k\Delta s)\,\Delta s$ is stored as the colors of vertices of all the polygons in the viewing pyramid. Finally, the second term is computed by rendering the shells with OpenGL's additive blending function. To render the shafts of light, the colors of the polygons have to be multiplied by attenuation ratio due to clouds, $\gamma(s)$. This can be easily achieved just by mapping the shadow texture onto the polygons using OpenGL's texture mapping function (see Fig. 7). The function can map the texture, multiplying the polygon's color by the values stored in the shadow texture. Since we have assumed $\gamma(s)$ is 1 above the cloud bottom, z_c, only the polygons under the cloud bottom need this mapping process.

The second step in the previous section has to be modified to render the shafts of light together with clouds. Since both of the billboards and the shells are transparent objects, they have to be rendered in back-to-front order. The procedure is as follows.

1. Calculate the colors of vertices of the polygons of the shells in the viewing pyramid.
2. Repeat the following steps for $k = n_s, n_s\text{-}1, \ldots, 1$.
   2.1 Render the shell k with additive blending function. Map the shadow texture for polygons under the cloud bottom.
   2.2 Render billboards for displaying clouds between the shell k-1 and the shell k.

6. RESULTS

We have made an animation to demonstrate the usefulness of our method. Figs. 8 through 11 show sequences of images from the animation.

Figs. 8 and 9 show the simulation of cloud formation. The number of cells is 256x128x20. The memory for store the binary distribution is 80 KB. In Fig. 8, clouds are formed above the mountains. Ellipsoids described in section 4.5 are randomly generated. Probabilities for vapor, phase transition and extinction are set to 0.1, 0.001, and 0.1, respectively, at the centers of ellipsoids. Images at every 50 steps are shown. The viewpoint is placed above clouds. Fig. 9 shows clouds formed around the mountain. In this example, ellipsoids are manually placed around the top of the mountain as an initial state. Vapor probability, phase transition probability, and extinction probability are the

same as those of Fig. 8 although they are forced to be 0 inside the mountain. These examples show the formation of clouds taking into account obstacles.

Figs. 10 and 11 show the effect of shafts of light. The cloud evolution is simulated on 256x256x20 cells. The memory for storing the binary distribution is 160 KB. Fig. 10 shows examples in daytime, and Fig. 11 is in the evening. The color of the sky is calculated by using the method proposed by Dobashi et al. [2]. In Fig. 10, shadows on the ground and mountains are also visible. The number of the spherical shells to calculate shafts of light is 40. As shown in these examples, the shafts of light have a strong visual impact to enhance the reality. In particular, the cloud color turns red in the evening and this results in a fascinating animation when combined with their movement.

The calculation was done on Intergraph TDZ 2000 GX1 (PentiumIII 500MHz Dual). In Figs. 8 and 9, the computation time for the simulation takes 0.3 seconds per one time step on average. In Figs. 10 and 11, the simulation takes 0.5 seconds per one time step. Images are rendered using the same machine. The image sizes are 640x480. The computation time for each image in Fig. 8 was less than 10 seconds. For Fig. 9, it took about 20 seconds. For images in Figs. 10 and 11, the computation time ranged from 20 to 30 seconds. These results indicate that the proposed method realizes the interactive simulation and fast generation of realistic images.

7. CONCLUSION

In this paper, we have proposed a realistic animation method for clouds. The cloud motion is simulated using the cellular automation. Realistic images including the shadows and the shafts of light are generated using OpenGL. Our method has the following advantages.

(1) Simulation of the cloud evolution requires only a small amount of computation since it is executed by Boolean operations.
(2) The memory requirement of the simulation is also small.
(3) Images can be rendered quickly by making use of graphics hardware.
(4) Shadows of clouds and shafts of light can also be rendered.

There remain a few things to be done in the future. First, the simulation should take into account the effects of terrain under clouds. In this case, the wind no longer blows in one direction; hence, our simulation method must be capable of handling multiple wind directions, or velocity fields. For faster image generation, our rendering method should include the idea of the level of detail. One possible approach is to represent the voxels hierarchically and to use coarser voxels in the distant regions from the viewer. This could reduce the computation time of the splatting process since the number of metaballs could be decreased.

APPENDIX

A. Implementation of Transition Rules

Let us assume the simulation space is $n_x \times n_y \times n_z = (m \times n) \times n_y \times n_z$ for simplicity. n is the bit length and m is an integer value. The necessary size of an array to store state variables is $m \times n_y \times n_z$. Let us denote the arrays for *hum*, *act*, *cld* at time step t_i as h[m][n_y][n_z][t_i], a[m][n_y][n_z][t_i], c[m][n_y][n_z][t_i], respectively. Rules expressed by Eqs. 1 and 2 can be executed in a straightforward way:

$$h[i][j][k][t_{i+1}] = h[i][j][k][t_i] \wedge \neg a[i][j][k][t_i], \quad (A.1)$$

$$c[i][j][k][t_{i+1}] = c[i][j][k][t_i] \vee a[i][j][k][t_i], \quad (A.2)$$

(i=0, …, m; j=0, …, n_y; k=0, …, n_z).

For the rule of Eq. 3, arrays that store bit patterns of *act* shifted in one and two bit are prepared. Let us denote them as a_right1, a_right2, a_left1, a_left2, respectively. Then Eq. 3 is calculated by the following equation.

$$\begin{aligned} a[i][j][k][t_{i+1}] = \neg a[i][j][k][t_i] \wedge h[i][j][k][t_i] \wedge \\ (a\_left1[i][j][k] \vee a\_left2[i][j][k] \vee a\_right1[i][j][k] \vee \\ a\_right2[i][j][k] \vee a[i][j+1][k][t_i] \vee a[i][j+2][k][t_i] \vee \\ a[i][j-1][k][t_i] \vee a[i][j-2][k][t_i] \vee a[i][j][k+1][t_i] \vee \\ a[i][j][k-1][t_i] \vee a[i][j][k-2][t_i]), \end{aligned} \quad (A.3)$$

(i=0, …, s; j=0, …, n_y; k=0, …, n_z).

To execute the rules of Eqs. 5 throught 7, n_t bit-sequences are prepared by setting each bit to 0 or 1 using random numbers that obey the probability i/n_p (i=0,..,n_p). The sequences are stored in the look-up table, p[i][j] (i=0,…,n_p; j=1,…,n_t). Using the table, the rules are executed as follows:

$$c[i][j][k][t_{i+1}] = c[i][j][k][t_i] \wedge p[\mathrm{int}(p_{ext} \times n_p)][\mathrm{int}(rnd \times n_t)], \quad (A.4)$$

$$h[i][j][k][t_{i+1}] = c[i][j][k][t_i] \wedge p[\mathrm{int}(p_{hum} \times n_p)][\mathrm{int}(rndz \times n_t)], \quad (A.5)$$

$$a[i][j][k][t_{i+1}] = c[i][j][k][t_i] \wedge p[\mathrm{int}(p_{act} \times n_p)][\mathrm{int}(rnd \times n_t)], \quad (A.6)$$

where int(x) indicates integer parts of x.

B. Field Function of Metaball as Basis Function

The field function proposed by Wyvill et al. is used [36]. The function is given by:

$$h(r) = \begin{cases} -\frac{4}{9}a^6 + \frac{17}{9}a^4 - \frac{22}{9}a^2 + 1, & (r \le R) \\ 0, & (r > R) \end{cases}, \quad (B.1)$$

where $a = r/R$, r is the distance from the center of a metaball to a calculation point, and R the effective radius of the metaball. To use the function as basis function, we normalize it by its total density. That is, the basis function is given by $f = h(r)/c$, where c is the normalizing factor. c is given by the following equation.

$$c = 4\pi \int_0^R h(r)dr = \frac{748}{405}\pi R . \quad (B.2)$$

C. Rendering Algorithm

Pseudo-code for the rendering clouds including the shafts of light is given here. In each element of the billboard texture, RGB components store the cumulative density and A component stores the attenuation ratio of the light passing through the metaball.

```
PROCEDURE DisplayImage()
 Place the camera at the sun position.
 Set the parallel projection.
 Clear screen with RGBA = (1.0, 1.0, 1.0, 1.0).
 ShadeClouds()
 Place the camera at the viewpoint.
 Set the perspective projection.
 Clear screen with background color.
 RenderObject() /* Rendering objects except clouds */
 RenderClouds()
END PROCEDURE

PROCEDURE ShadeClouds()
 Sort metaballs in ascending order from the sun.
 glDisable(GL_DEPTH_TEST)
 glBlendFunc(GL_ZERO, GL_SRC_ALPHA)
 glEnable(GL_BLEND)
 FOR k = each metaball DO
```

Place the billboard at the center of metaball *k*.
Rotate the billboard so that its normal is oriented to the sun.
Set the billboard color as RGBA = (1.0, 1.0, 1.0, 1.0).
Map the billboard texture with GL_MODULATE.
Render the billboard.
Read the pixel value corresponding to the center of metaball *k.*
Multiply the pixel value by the sunlight color.
Store the color into an array C[*k*] as the color of the billboard.
END FOR
Store the image (T) in the frame buffer as a light map texture.
END PROCEDURE

PROCEDURE RenderClouds()
Sort metaballs in descending order from the viewpoint.
Store distances between metaballls and the viewpoint in an array D.
glDisable(GL_DEPTH_TEST)
glEnable(GL_BLEND)
k = number of shells
WHILE D[0] < distance to shell *k* **DO**
RenderShell(*k*)
k = k – 1
END WHILE
glBlendFunc(GL_ONE, GL_SRC_ALPHA)
FOR *n* = each metaball **DO**
IF D[*n*] < distance to shell *k* **DO**
Render shell(*k*)
k = k – 1
glBlendFunc(GL_ONE, GL_SRC_ALPHA)
END IF
Place the billboard at the center of the corresponding metaball *n.*
Rotate the billboard so that its normal is oriented to the viewpoint.
Set the billboard color as C[*n*].
Map the billboard texture.
Render the billboard with the blending function.
END FOR
WHILE *k* > 0 **DO**
RenderShell(*k*)
k = k – 1
END WHILE
END PROCEDURE

PROCEDURE RenderShell(*k*)
Calculate colors of vertices of shell *k.*
Map the light map texture T with GL_MODULATE.
Render shell *k.*
glBlendFunc(GL_ONE, GL_ONE)
END PROCEDURE

REFERENCES

[1] D. Blythe, "Advanced Graphics Programming Techniques Using OpenGL," *Course Note #29 of SIGGRAPH 99*, 1999.

[2] Y. Dobashi, T. Nishita, K. Kaneda, H. Yamashita, "A Fast Display Method of Sky Color Using Basis Functions," *The Journal of Visualization and Computer Graphics*, Vol. 8, No. 2, 1997, pp. 115-127.

[3] Y. Dobashi, T. Nishita, T. Okita, "Animation of Clouds Using Cellular Automaton," *Proc. of Computer Graphics and Imaging'98*, 1998, pp. 251-256.

[4] Y. Dobashi, T. Nishita, H. Yamashita, T. Okita, "Using Metaballs to Modeling and Animate Clouds from Satellite Images," *The Visual Computer*, Vol. 15, No. 9, 1998, pp. 471-482.

[5] D. S. Ebert, R. E. Parent, "Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-Buffer Techniques," *Computer Graphics*, Vol. 24, No. 4, 1990, pp. 357-366.

[6] D. S. Ebert, W. E. Carlson, R. E. Parent, "Solid Spaces and Inverse Particle Systems for Controlling the Animation of Gases and Fluids," *The Visual Computer*, 10, 1990, pp. 471-483.

[7] D. S. Ebert, "Volumetric Modeling with Implicit Functions: A Cloud is Born," *Visual Proc. of SIGGRAPH'97*, 1997, pp. 147.

[8] D. S. Ebert, "Simulating Nature: From Theory to Application," *Course Note #26 of SIGGRAPH 99*, 1999, pp. 5.1-5.52.

[9] N. Foster, D. Metaxas, "Modeling the Motion of a Hot, Turbulent Gas," *Proc. of SIGGRAPH'97*, 1997, pp. 181-188.

[10] G.Y. Gardner, "Visual Simulation of Clouds," *Computer Graphics*, Vol.19, No. 3, 1985, pp. 279-303.

[11] M. Inakage, "Volume Tracing of Atmospheric Environments, " *The Visual Computer*, 7, 1991, pp. 104-113.

[12] H. W. Jansen, P. H. Christensen, "Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps," *Proc. of SIGGRAPH'98*, 1998, pp. 311-320.

[13] J. T. Kajiya, B. P. V. Herzen, "Ray Tracing Volume Densities," *Computer Graphics*, 1984, Vol. 18, No. 3, pp. 165-174.

[14] K. Kaneda, T. Okamoto, E. Nakamae, T. Nishita, "Photorealistic Image Synthesis for Outdoor Scenery under Various Atmospheric Conditions," *The Visual Computer*, 7(5&6), 1991, pp. 247-258.

[15] T. Kikuchi, K. Muraoka, and N. Chiba, "Visual Simulation of Cumulonimbus Clouds," *The Journal of The Institute of Image Electronics and Electronics Engineers of Japan*, Vol. 27, No. 4, 1998, pp. 317-326 (in Japanese).

[16] N. Max, "Light Diffusion through Clouds and Haze," *Graphics and Image Processing*, Vol. 13, No. 3, 1986, pp. 280-292.

[17] N. Max, "Atmospheric Illumination and Shadows," *Computer Graphics*, Vol. 20, No. 4, 1986, pp. 117-124.

[18] N. Max, R. Crawfis, D. Williams, "Visualizing Wind Velocities by Advecting Cloud Textures," *Proc. of Visualization'92*, 1992, pp. 179-183.

[19] N. Max, "Efficient Light Propagation for Multiple Anisotropic Volume Scattering," *Proc. of the Fifth Eurographics Workshop on Rendering*, 1994, pp. 87-104.

[20] K. Meuller, N. Shareef, J. Huang, R. Crawfis, "Hight-Quality Splatting on Rectilinear Grids with Efficient Culling of Occluded Voxels," *IEEE Trans. on Visualization and Computer Graphics*, Vol. 5, No. 2, 1999, pp. 116-134.

[21] K. Nagel, E. Raschke, "Self-Organizing Criticality in Cloud Formation?," *Physica A*, 182, 1992, pp. 519-531.

[22] F. Neyret, "Qualitative Simulation of Convective Clouds Formation and Evolution," *Proc of Eurographics Computer Animation and Simulation Workshop'97*, 1997, pp. 113-124.

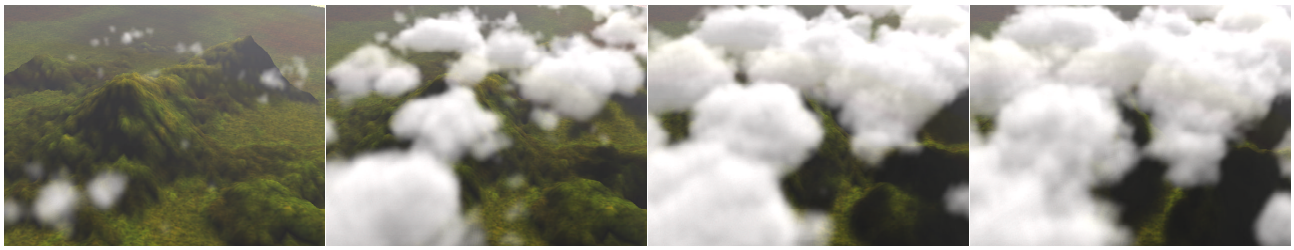
[23] T. Nishita, Y. Miyawaki, E. Nakamae, "A Shading Model for Atmospheric Scattering Considering Distribution of Light Sources," *Computer Graphics*, Vol. 21, No. 4, 1987, pp. 303-310.

[24] T. Nishita, E. Nakamae, "Method of Displaying Optical Effects within Water using Accumulation Buffer," *Proc. of SIGGRAPH'94*, 1994, pp. 373-379.

[25] T. Nishita, Y. Dobashi, E. Nakamae, "Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light," *Proc. of SIGGRAPH'96*, 1996, pp. 379-386.

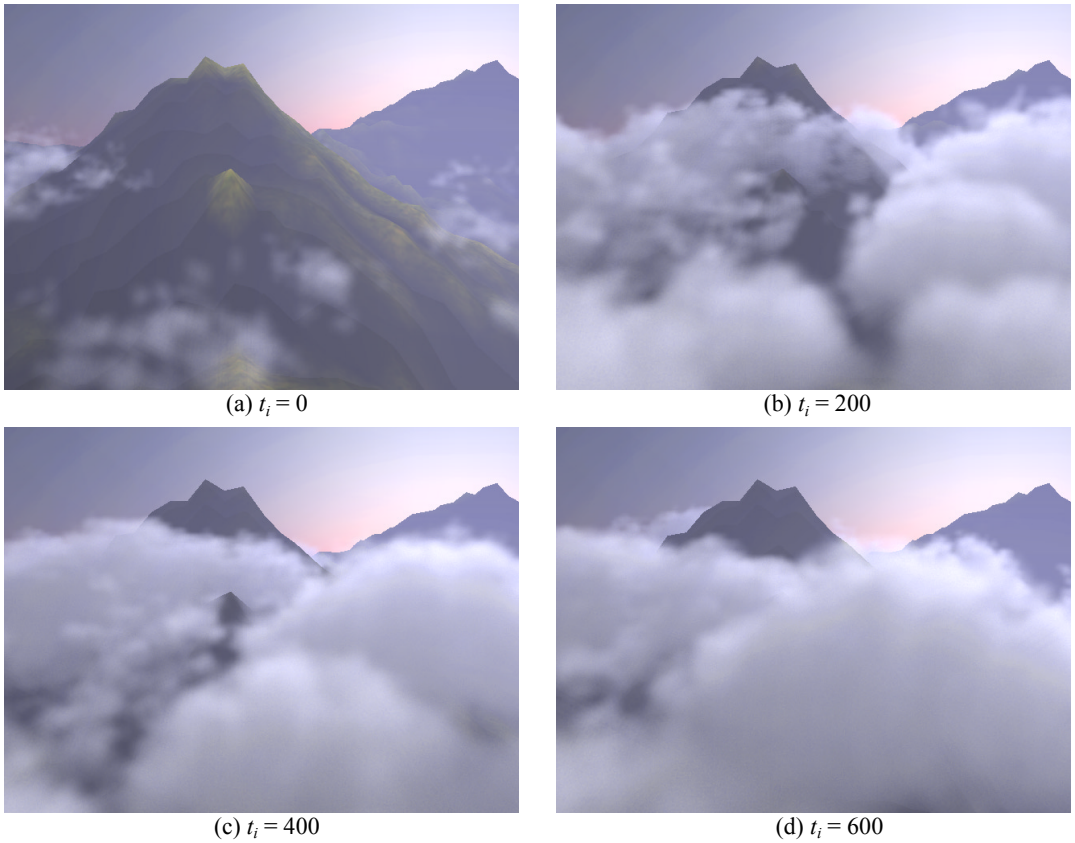
[26] A. J. Preetham, P. Shirley, B. Smits, "A Practical Analytic Model for Daylight," *Proc. of SIGGRAPH'99*, 1999, pp. 91-100.

[27] H. E. Rushmeier, K. E. Torrance, "The Zonal Method for Calculating Light Intensities in The Presence of a Participating Medium," *Computer Graphics*, Vol. 21, No. 4, 1987, pp. 293-302.
[28] G. Sakas, M. Gerth, "Sampling and Anti-Aliasing of Discrete 3-D Volume Density Textures," *Proc. of EUROGRAPHICS'91*, 1991, pp. 87-102.
[29] J. Stam, E. Fiume, "Turbulent Wind Fields for Gaseous Phenomena," *Proc. of SIGGRAPH'93*, 1993, pp. 369-376.
[30] J. Stam, "Stochastic Rendering of Density Fields," *Proc. of Graphics Interface'94*, 1994, pp. 51-58.
[31] J. Stam, E. Fiume, "Dipicting Fire and Other Gaseous Phenomena Using Diffusion Processes," *Proc. of SIGGRAPH'95*, 1995, pp. 129-136.
[32] J. Stam, "Stable Fluids," *Proc. of SIGGRAPH'99*, 1999, pp. 121-128.
[33] R. Voss, "Fourier Synthesis of Gaussian Fractals: 1/*f* noises, landscapes, and flakes," *SIGGRAPH'83: Tutorial on State of the Art Image Synthesis*, 10, 1983.
[34] L. Westover, "Footprint Evaluation for Volume Rendering," *Computer Graphics*, Vol. 24, No. 4, 1990, pp. 367-376.
[35] S. Wolfram, "Cellular automata as models of complexity," *Nature*, Vol. 311, No. 4, 1984, pp. 419-424.
[36] G. Wyvill, A. Trotman, "Ray-Tracing Soft Objects," *Proc. of CG International*, 1990, pp. 439-475.



(a) $t_i = 0$ (b) $t_i = 50$ (c) $t_i = 100$ (d) $t_i = 150$

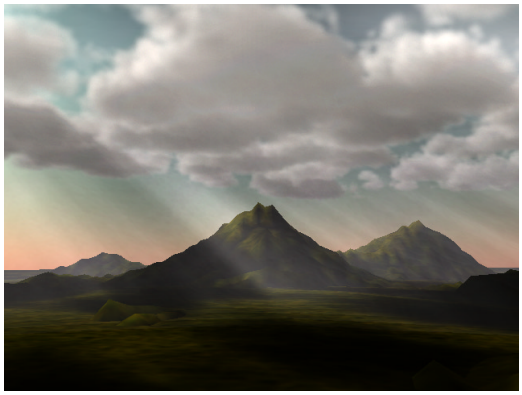
Figure 8: Simulation of cloud formation.



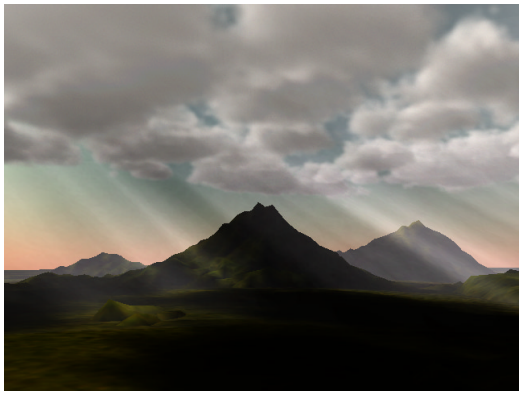
(a) $t_i = 0$ (b) $t_i = 200$

(c) $t_i = 400$ (d) $t_i = 600$

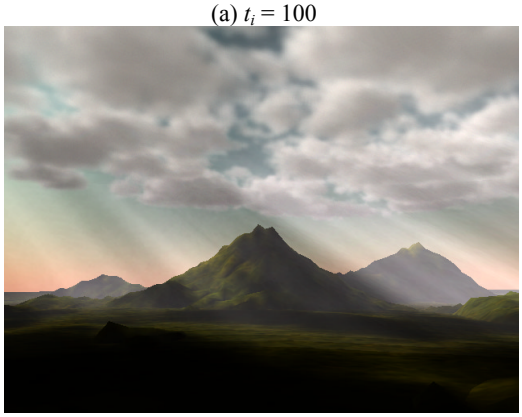
Figure 9: Cloud formation around mountains.



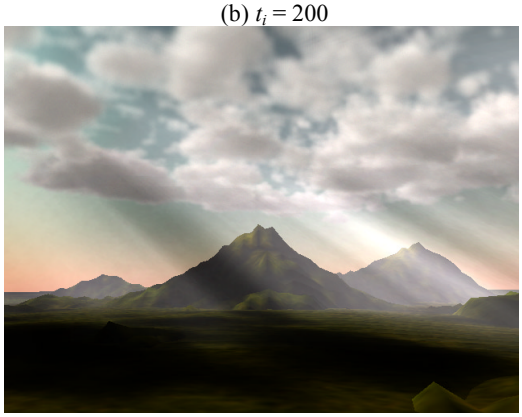
(a) $t_i = 100$



(b) $t_i = 200$

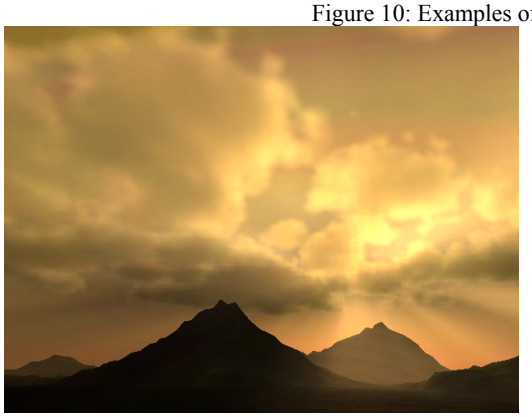


(c) $t_i = 300$

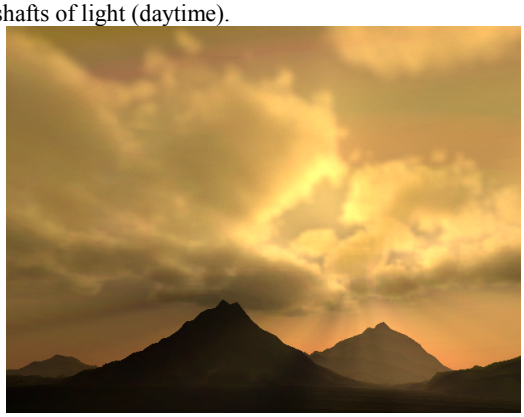


(d) $t_i = 400$

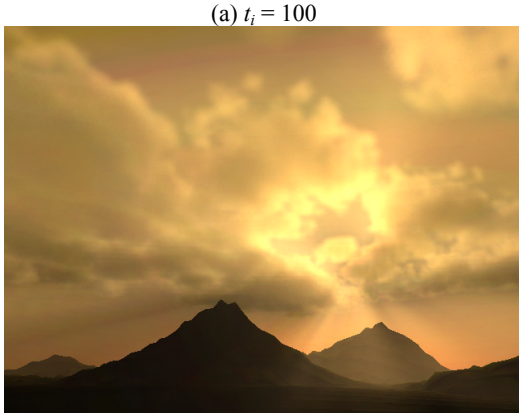
Figure 10: Examples of shafts of light (daytime).



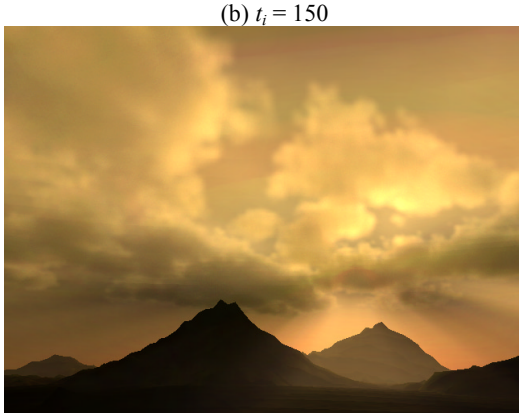
(a) $t_i = 100$



(b) $t_i = 150$



(c) $t_i = 200$



(d) $t_i = 250$

Figure 11: Examples of shafts of light (evening).