Efficient Rendering of Lightning Taking into Account Scattering Effects due to Clouds and Atmospheric Particles

Yoshinori Dobashi Tsuyoshi Yamamoto Tomoyuki Nishita[†]

Hokkaido University
Kita 13, Nishi 8, Kita-ku,
Sapporo, 060-8628, Japan
{doba, yamamoto}@nis-ei.eng.hokudai.ac.jp

[†]University of Tokyo
7-3-1, Hongo, Bunkyo-ku
Tokyo, 113-0033, Japan
nis@is.s.u-tokyo.ac.jp

Abstract

A number of methods have been developed for creating realistic images of natural scenes. Their applications include flight simulators, the visual assessment of outdoor scenery, etc. Previously, many of these methods have focused on creating images under clear or slightly cloudy days. Simulations under bad weather conditions, however, are one of the important issues for realism. Lightning is one of the essential elements for these types of simulations. This paper proposes an efficient method for creating realistic images of scenes including lightning. Our method can create photo-realistic images by taking into account the scattering effects due to clouds and atmospheric particles illuminated by lightning. Moreover, graphics hardware is utilized to accelerate the image generation. The usefulness of our method is demonstrated by creating images of outdoor scenes that include lightning.

Keywords: Lightning, Clouds, Atmospheric Scattering, Efficient Rendering, Level of Detail, Graphics Hardware

1. Introduction

The simulation of natural phenomena using computer graphics, such as clouds, water, terrain, smoke or fire, is one of the most important research areas. A number of methods have been developed and there exist many applications that use these methods, such as flight simulators and the visual assessment of outdoor scenes. For these applications, simulations not only under fine weather conditions but also under bad weather conditions are very important. There are many important elements for creating realistic images of scenes under bad weather conditions. For example, the effects of windstorm, sand storms, rain and lightning should be taken into account. Amongst these scenes, this paper focuses on rendering images that include lightning. Several methods have been developed for rendering images of lightning [1][2][3] that are employed by a lot of commercial software. These previous methods, however, focus on the rendering of the lightning itself. However, the flash of lightning illuminates the surrounding clouds and atmospheric particles. That is, the clouds are brightened and there is a glow surrounding lightning strokes. These phenomena must be taken into account to create realistic images. The previous methods display these phenomena by using heuristic algorithms and their models are different from the actual physical phenomena. Physical-based rendering is desirable for realistic image synthesis. Moreover, most of the previous methods employ the ray-tracing algorithm for image generation. This results in increasing computation time.

This paper proposes a new method to address these problems. Our method can create realistic images by taking into account not only the lightning itself, but also scattering effects due to cloud and atmospheric particles illuminated by the lightning. The clouds and the effects of atmospheric scattering are rendered based on the physical phenomena. Furthermore, graphics hardware is utilized to accelerate the computation. Recently, several hardware-accelerated methods for rendering clouds and the atmospheric effects have been proposed [4][5][6][7]. Unfortunately, to our knowledge, there seems to be no hardware-accelerated method, at present, for rendering images that include lightning. Using our method, realistic scenes that include lightning can be generated within half a minute. Since computers and graphics hardware are becoming faster and faster, we believe that real-time rendering using our method will be realized in the near future.

In this paper, Section 2 discusses the previous work

related to the rendering of lightning. Next, in Section 3, the basic idea of our proposed method is described. In Section 4, methods for efficient calculation of the intensity of color of clouds and atmospheric scattering of light due to lightning are proposed. Then, the usefulness of the proposed method is demonstrated by several examples in Section 5. Finally, conclusions and future work are discussed in Section 6.

2. Related Work

In the field of computer graphics, the first method for creating synthetic images of lightning was developed by Reed et al. [1]. In their method, the strokes of lightning are probabilistically generated by using particle systems. The lightning and its glow are rendered by employing a heuristic approach. However, clouds illuminated by lightning are not taken into account, since the purpose of their method is solely for creating images of lightning. Moreover, their model for the glow is completely different from the actual physical phenomena, since one of the physical mechanisms for the glow is atmospheric scattering due to the flash of lightning. This must be taken into account to create realistic images. Kruszenwski proposed an alternative method for generating the lightning strokes based on probabilistic theory [2]. This method can create the desirable shape of lightning interactively by controlling several parameters. The novelty of his method is in the modeling. He employs the same method as [1] for rendering. So, the scattering effects due to cloud and atmospheric particles are not taken into account. Sosorbaram et al. developed a method for simulating lightning taking into account clouds [3]. In the paper, a method for rendering clouds illuminated by lightning was proposed. They use the ray-tracing algorithm. Therefore, their method requires a great deal of time.

There have been a large number of research reports on the generation of images which take into account scattering effects due to the presence of clouds and atmospheric particles [8][9][10][11][12][13][14][15][16]. In these methods, light sources are assumed to be natural light sources, such as sunlight/skylight, and/or artificial light sources such as spotlights. So, they are not suitable for our purpose where the lightning strokes themselves are the light sources. Moreover, most of them require dozens of minutes to create an image, since they use the ray-tracing algorithm. Recently, hardware-accelerated methods have been proposed for fast image generation. Stam has used a three-dimensional hardware texture mapping function to display smoke in real-time [4]. However, this method focuses on simulating the motion of smoke and the atmospheric scattering is not taken into account. Dobashi et al. have developed an efficient method for rendering clouds together with the atmospheric scattering [5]. This method is limited to an infinite light source such as sunlight. They have been able to extend this to point light sources [6]. The purpose of this method, however, is to render shafts of light due to spotlights and is not applicable to clouds. Therefore, the previous methods are not suitable for our purpose of rendering clouds illuminated by lightning and to render its glow taking into account the atmospheric scattering.

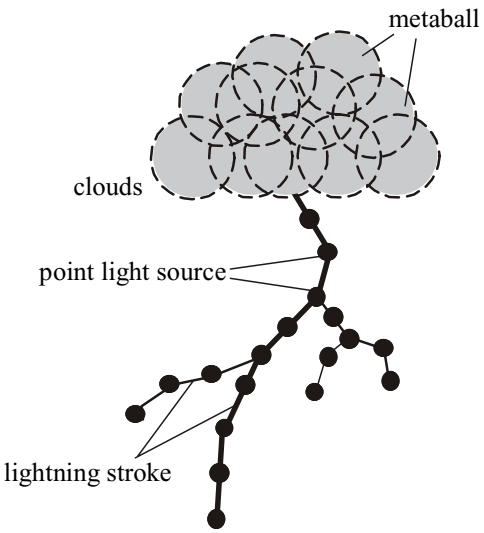


Figure 1. Basic idea of our method. Point light sources are placed on lightning strokes to compute the intensity of clouds illuminated by lightning and the glow of lighting due to atmospheric scattering.

3. Basic Idea of Our Method

The lightning strokes are generated by using Reed's method [1], and are represented by a set of line segments. That is, an initial segment in the clouds is generated at first. Then new segments are repeatedly spawned at the endpoints of the previous segments. The lengths and the orientations of the new segments are determined randomly [1]. Lightning is displayed by a line drawing function of OpenGL. The color of the lightning is specified by the user. In general, lightning consists of a main channel and several branches. The widths and intensities of the lightning strokes are decreased as they branch. These are simulated by decreasing the widths and intensities of the line segments according to a specified ratio.

The intensity of the clouds illuminated by lightning and of the atmospheric scattering due to lightning are computed by generating point light sources on the line segments as shown in Fig. 1. Then, the clouds and atmospheric scattering are rendered by summing contributions from each point light source. We assume the

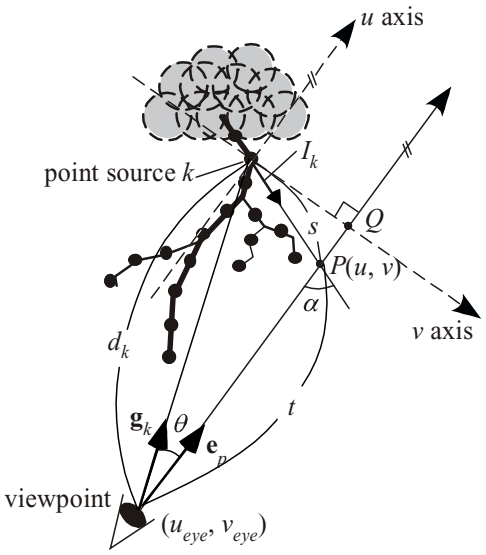


Figure 2. Computation of atmospheric scattering due to lightning.

density distribution of the atmospheric particles is uniform. This makes it possible to prepare a lookup table in a pre-process that stores the intensity of scattered light due to a single point source. The total intensity of light reaching the viewpoint can be quickly obtained by using this table. The density distribution of clouds is expressed by a set of metaballs [14][5]. Clouds illuminated by lightning are rendered efficiently by making use of graphics hardware. We extend the method developed by Dobashi et al. [5] to allow it to handle point light sources for rendering clouds. Moreover, to realize further speed improvements, we apply the idea of level of detail (LOD) to clouds by using octrees.

4. Efficient Rendering of Scenes with Lightning

In this section, an efficient method for calculating the intensity of light due to atmospheric scattering is described. Then, the intensity of clouds illuminated by lightning is explained.

4.1 Rendering of Atmospheric Scattering due to Lightning

Fig. 2 shows the idea of our method. Let us denote the intensity of a point source, k, on the lightning strokes as $I_k(\lambda)$, where λ is a wavelength. When light from the point source k reaches point P on the viewing ray, it is scattered by the particles and then reaches the viewpoint. Let us assume that the density distribution of the atmospheric particle is uniform. Then, the intensity of light reaching the viewpoint, $I_s^{(k)}(\lambda)$, is given by the following equation [11].

$$I_s^{(k)}(\lambda) = \rho_a F_\lambda(\cos\alpha) \times \frac{\exp(-\rho_a \kappa_a (s+t))}{s^2} I_k(\lambda), \quad (1)$$

where ρ_a is the density of the particles, F_λ the phase function of the particles, α the phase angle (see Fig. 2), κ_a the extinction coefficient, s the distance between point P and point source k and t is the distance between point P and the viewpoint. The phase function F_λ is typically expressed as a cosine function of the phase angle α as shown in Eq. 1. The total intensity of light scattering due to the point source k, $I_{eye}^{(k)}(\lambda)$, is obtained by integrating $I_s^{(k)}(\lambda)$ along with the viewing ray. That is,

$$I_{eye}^{(k)}(\lambda) = \int_0^\infty \rho_a F_\lambda(\cos\alpha) \times \frac{\exp(-\rho_a \kappa_a (s+t))}{s^2} I_k(\lambda) dt. \quad (2)$$

To calculate $I_{eye}^{(k)}(\lambda)$, $I_s^{(k)}(\lambda)$ has to be integrated along the viewing ray from the viewpoint to an infinite point as expressed in Eq. 2. However, this is practically impossible, so instead we truncate the integration by a large value, T, specified by the user. The intensity of each pixel is obtained by calculating $I_{eye}^{(k)}(\lambda)$ for all the light sources on the lightning strokes and summing them. Since no analytical solutions are available for Eq. 2, it must be calculated numerically by generating sample points on the viewing ray. However, this is very time-consuming. To address this problem, the proposed method pre-calculates $I_{eye}^{(k)}(\lambda)$ and stores these values in a lookup table. The details are described in the following.

As shown in Fig. 2, let us define a local coordinate system with its origin at the position of the point source. In this coordinate system, the u axis is parallel to the viewing ray and passes through the position of the point source. The v axis is defined by a line connecting the point source k with the point Q (see Fig. 2). The point Q is on the viewing ray and is the closest point to the point source k. Let the coordinates of point P and the viewpoint in this system be (u, v) and (u_{eye}, v_{eye}), respectively. The following equation is then obtained.

$$\begin{cases} s = \sqrt{u^2 + v^2} \\ t = u - t_{eye} \\ \cos\alpha = -u / \sqrt{u^2 + v^2} \end{cases} \quad (3)$$

Substituting Eq. 3 into Eq. 2, the following equation is obtained.

$$I_{eye}^{(k)}(\lambda) = I_k I_l(u_{eye}, v_{eye}, \lambda), \quad (4)$$

where $I_l(u_{eye}, v_{eye}, \lambda)$ is given by:

$$I_l(u_{eye}, v_{eye}, \lambda) = \int_{u_{eye}}^{T} \rho_a F_\lambda\left(\frac{-u}{\sqrt{u^2+v^2}}\right) \times \frac{\exp(-\rho_a \kappa(\sqrt{u^2+v^2}+u-u_{eye}))}{u^2+v^2} du \quad (5)$$

As shown in Eq. 5, $I_l(u_{eye}, v_{eye}, \lambda)$ can be calculated numerically, given the local coordinate of the viewpoint (u_{eye}, v_{eye}). Therefore, in the proposed method, $I_l(u_{eye}, v_{eye}, \lambda)$ is calculated in advance by changing u_{eye} and v_{eye} from $-T$ to T. The value is stored in a two-dimensional lookup table. Note that the wavelength λ is sampled at 675, 520, and 460 [nm]. These values correspond to RGB components. Using this table, the intensity of the pixel p, $I_p(\lambda)$, is obtained by the following equation.

$$I_p(\lambda) = \sum_{k=1}^{n} I_k I_l(u_{p,k}, v_{p,k}, \lambda)\Delta l, \quad (6)$$

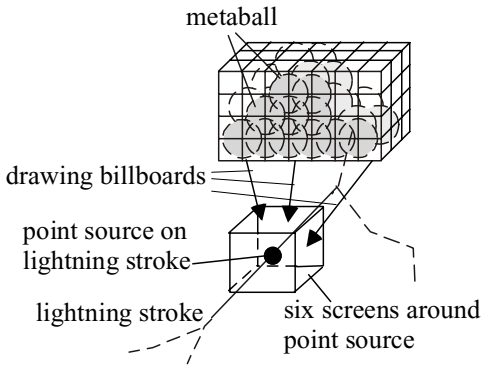
where n is the number of point sources on the lightning strokes, Δl the distance between adjacent point sources and $(u_{p,k}, v_{p,k})$ are the local coordinates of the viewpoint determined by the position of point source k and the viewing ray of pixel p. $(u_{p,k}, v_{p,k})$ is easily obtained by the following equations.

$$\begin{cases} u_{p,k} = -d_k \cos\theta = -d_k(\mathbf{g}_k \bullet \mathbf{e}_p) \\ v_{p,k} = \sqrt{d_k^2 - u_{p,k}^2} \end{cases}, \quad (7)$$

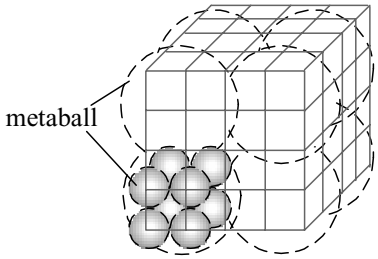
where d_k is the distance between the viewpoint and point source k, $\mathbf{g}_k$ the unit vector toward the direction of the point source k viewed from the viewpoint, $\mathbf{e}_p$ the unit vector toward the viewing direction at pixel p, and θ the angle between $\mathbf{g}_k$ and $\mathbf{e}_p$ (see Fig. 2).

4.2 Fast Calculation for Rendering Clouds Illuminated by Lightning Using LOD

In this subsection, the calculation method for rendering clouds illuminated by lightning is described. The density distribution of clouds is defined on three-dimensional voxels as shown in Fig. 3(a). The density of each voxel is determined by using the method described in [5]. The color of clouds is calculated by extending the method in [5]. That is, metaballs are placed at the center of each voxel and graphics hardware is utilized to accelerate the computation. A metaball is a sphere and a field function is defined inside it [17]. The metaball has two parameters, that is, the center density q and the radius of the sphere, R. R is called an effective radius. The center density of each metaball is set to the density of clouds at the voxel. The effective radius is specified by the user. The intensity of color of the clouds is calculated by summing the intensity of light reaching the metaballs from each point light source on the lightning strokes. Using graphics hardware, the intensity of light reaching the metaball is efficiently



(a) Calculation of cloud color



(b) Hierarchical representation of clouds using octree

Figure 3. Efficient calculation of cloud color using LOD.

calculated as follows. A cube is placed at the point source as shown in Fig. 3(a). The center of the cube is at the position of the point source. The intensity of light reaching each metaball is computed by creating images of clouds viewed from the center of the cube by assuming each face of the cube acts as a screen. The images are generated by the splatting method using billboards [5]. First, the billboard (a square) is prepared and textures for the billboard are pre-calculated. Each element of the texture stores the attenuation ratio and the cumulative density of the light passing through the metaball. The textures are mapped onto the billboards. Next, the billboards are placed at the centers of the metaballs and the frame buffers are initialized to 1.0 (white). Then, the billboards are sorted in ascending order depending on their distances from the center of the cube and they are projected onto one of the six screens. The values in the frame buffer are multiplied by their attenuation ratios stored in the billboard texture. This process is done by using a hardware blending function. Then the pixel value corresponding to the center of the metaball is read back from the frame buffer. This value is equal to the attenuation ratio between the metaball and the center of the cube, i.e. the position of the point source. The intensity of light reaching the metaball is obtained by multiplying the attenuation ratio by the intensity of the

point source divided by the square of the distance between the metaball and the point source. After repeating these processes for all metaballs, the clouds viewed from the viewpoint are created by drawing the billboards in back to front order. For more details see [5].

Basically, the intensity of clouds illuminated by lightning can be calculated by using the above method. This method, however, requires long time to complete. The computational cost of the method is proportional to the number of metaballs and the point sources on the lightning strokes. In our experiment, thousands of metaballs are required to create realistic shapes of clouds and a lot of point sources have then to be placed on the lightning strokes to calculate the color of clouds accurately. In the example shown in Section 5, there are about a hundred thousand metaballs and fifty point sources. In this case, we have found that it takes several minutes to create a single image by using the above method. So, to address this problem, we make use of the idea of LOD.

The intensity of light from the point source decreases in proportion to the square of the distance. Therefore, a straightforward approach is to ignore the metaballs far from the point source, since the intensity of light reaching the metaballs is then considered to be very small. However, the contributions from a lot of point sources can become important, even if the contribution from one point source is very small. Therefore, this straightforward approach does not work well. Actually we found that the quality of images by this approach is not sufficient. Paquette et al. pointed out the similar problem and proposed an efficient method for shading objects illuminated by many point sources [18]. In the method, point sources are grouped into a hierarchical structure for the efficient computation. Unfortunately, this method is not applicable to our case, since their purpose is to calculate the illuminance on diffuse and/or specular surfaces. Moreover, the number of metaballs is far larger than that of the point sources on the lightning strokes. Therefore, we do not expect to reduce the rendering time drastically by using Paquette's method [18]. Instead, metaballs are grouped into the hierarchical structure in our method. The intensity of light from the point source is small at positions far from the point source and all the metaballs in the areas receive almost the same energy of light. So, the proposed method uses a larger metaball representing multiple metaballs in those areas for the intensity calculation. The details are explained in the following.

First, clouds are represented by the octree. As mentioned previously, the density distribution of clouds is defined on voxels. So, as shown in Fig. 3(b), eight neighboring voxels are grouped and replaced by a larger voxel. By repeating this process, the density distribution is represented by the octree. Let us consider a voxel at level l in the octree. Corresponding to the size of the voxel, larger metaballs are placed (see Fig.3(b)). In the following, the larger metaballs are called *parent metaballs* and the eight metaballs in the same group are called *child metaballs*. The center density of the parent metaball is set to the mean density of eight child metaballs and the effective radius is set to be twice as large as that of their child metaballs. For regions far from the point sources, the parent metaballs are used to calculate the intensity of light reaching from the point sources. The intensities of light reaching the child metaballs are set to the same value of their parent metaball. This reduces the computation time since the number of metaballs to be processed is decreased. The remaining issue is then the selection of the appropriate level l in the octree to satisfy a user-specified tolerance, ε.

The intensity of light, $I_{kj}(\lambda)$, reaching metaball j from point source k is calculated by the following equation.

$$I_{kj}(\lambda) = \frac{I_k(\lambda)\exp(-\tau(r))}{r^2}, \tag{9}$$

where r and τ are respectively the distance and the optical length between point source k and metaball j. The optical length is calculated by integrating the density of cloud particles between point source k and metaball j. That is,

$$\tau(r) = \kappa_c \int_0^r \rho_c(l)dl, \tag{10}$$

where κ_c and ρ_c are the extinction coefficients and the density of cloud particles, respectively. Let us consider an energy E_j of light received by metaball j. Approximately, E_j is given by multiplying the intensity I_{kj} by the volume of metaball j. $E_j = I_{kj}(\lambda)\mathrm{d}V_j$, where $\mathrm{d}V_j$ denotes the volume. One possible condition for selecting the appropriate level is as follows.

$$\frac{\max_{\lambda}\{I_k(\lambda)\exp(-\tau(r))\}}{r^2} < \varepsilon, \tag{11}$$

where ε is a user-specified tolerance. The evaluation of Condition (11) requires the computation of $\exp(-\tau(r))$. However, computing it is equal to using child metaballs to compute the intensity of light reaching themselves. This does not reduce the computation time. Therefore, instead, we use the following equation.

$$\frac{\max_{\lambda}\{I_k(\lambda)\}}{r^2} < \varepsilon \tag{12}$$

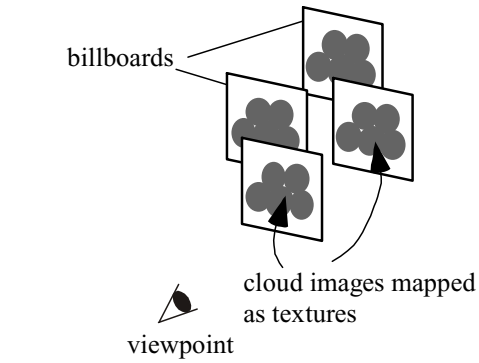
The left side of Condition (12) indicates the energy that is received by metaball j when there are no clouds between the metaball and the light source. The computational cost of Condition (12) is very small since it only requires the distance between the metaball and the light source. The coarsest level l that satisfies Condition (12) is searched in the octree and the obtained metaball is used for the intensity calculation of light reaching from the point source.

4.3 Fast Rendering of Clouds Using Hierarchical Imposters

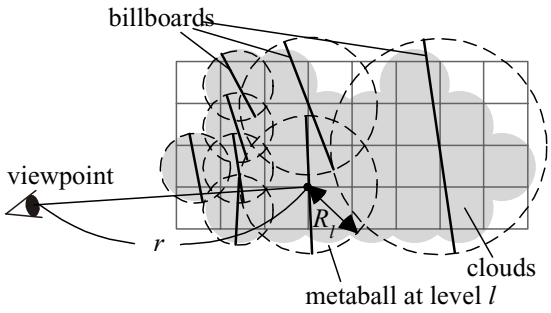
Fast image generation is indispensable for applying the proposed method to flight simulations under bad weather conditions. In this subsection, an efficient method for rendering clouds is proposed. We employ imposters to achieve this end. An imposter replaces an object with a semi-transparent polygon texture-mapped with an image of the object it replaces. The image is a rendering of the object viewed through the polygon. In our case, multiple metaballs are replaced with the polygon as shown in Fig. 4(a). The texture mapped onto the polygon is the image of the metaballs. This texture can be reused within some error tolerance as long as the viewpoint is near the point where the texture is generated. Therefore, this makes it possible to reduce the rendering time for a flythrough animation. There are several methods using the idea of the imposters [19][20][21]. The purpose in these methods, however, is to realize a real-time walkthrough of scenes consisting of numbers of polygons such as buildings and terrains. Their applications to clouds has not been demonstrated. So, Harris et al. developed a method using the imposters for fast rendering of clouds. In this method, clouds are divided into numbers of clusters and the imposter textures are generated for each cluster. In their method, the clustering of clouds is static and imposters for clouds are always used no matter where they are in relation to the viewer. However, the precise rendering of clouds is not realized using this method when the viewpoint is close to them. Clouds should be divided into appropriate clusters depending on the distance from the viewpoint.

To address the problem, our method uses imposters in a hierarchical manner. We use the octree described in the previous section. As shown in Fig. 4(b), our system selects appropriate level l in the octree depending on the distance from the viewpoint. Coarser voxels are selected in the distant regions, and finer voxels in the close regions. Note that Fig. 4(b) depicts the two-dimensional case for simplicity. Then, as shown in Fig. 4(c), an image of the metaballs in the selected voxel is generated by considering the billboard of the larger metaball as the screen. This image is stored as a texture and it is mapped onto the billboard. The texture-mapped billboard is used as the imposter to create the final image. The selection process of the level in the octree is as follows.

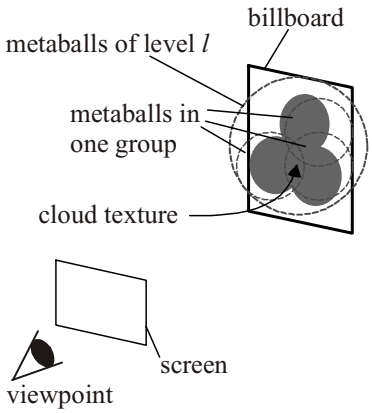
Let us denote the effective radius of the metaball at level l in the octree as R_l. The sphere with radius R_l is considered to be a bounding sphere of its child metaballs. When the viewpoint is far from the clouds, it is visually convincing to replace a lot of the metaballs with a single imposter. On the contrary, using too many metaballs for the imposter would result in unnatural images when the



(a) Basic concept of imposters



(b) Hierarchical generation of imposters



(c) Creating cloud texture

Figure 4. Fast rendering of clouds using hierarchical imposters.

viewpoint is close to them. Therefore, we determine the level l by the following equation using the radius R_l.

$$\frac{R_l}{r} < C_{dist}, \tag{13}$$

where C_{dist} is a user-specified tolerance and r is the distance from the viewpoint to the metaball (see Fig. 4(b)). The user can control the image quality and the rendering time by choice of C_{dist}. Smaller values of C_{dist} result in increasing the number of metaballs since finer levels are then selected. This method makes it possible to

create imposters adaptively depending on the distance from the viewpoint. However, creating the imposter textures at every frame requires the same cost as that of rendering all the metaballs. In practice, this results in an increase in the rendering time, since the textures then have to be transferred into texture memories. Then the textures are updated only when changes in the angles of the camera direction viewed from the metaballs have become greater than an angular tolerance, C_{ang}, specified by the user. In the distant regions, the textures are not updated frequently since a sudden change seldom occurs. On the other hand, the textures are often updated in the regions near the viewpoint. However, the textures can be quickly generated since the number of metaballs in the same group is small in the near regions.

In order to verify the effectiveness of our method, we have created a simple animation. In the animation, the camera approaches clouds after going around them. The two tolerance values for updating the imposter textures, C_{dist} and C_{ang}, are set to 0.1 and 5, respectively. In this animation, there were 30 frames and the animation takes 130 seconds without using the imposters. The rendering time is reduced to 30 seconds by using the imposters. The proposed method can create the animation four times faster.

5. Result

In this section, the usefulness of the proposed method is demonstrated by several examples. To create images shown in this section, the size of the lookup table for the atmospheric scattering is 128x128 (see Section 4.1). The value of T (see Eq. 5) for integrating intensity of atmospheric scattering to compute this table is set to 1.5 [km]. The intensity of clouds is efficiently calculated by using the octree (see Section 4.2). The tolerance, ε, for the intensity error is 0.2 (see Eq. 12).

First, the effectiveness of the proposed method is verified by using a simple example. Fig. 5 shows example images of scenes that include lightning. In the figure, there is one lightning. 50 point light sources are generated on the lightning strokes. The realistic glow due to scattering effects of atmospheric particles illuminated by the lightning is depicted. The clouds are brightened since they are illuminated by lightning. In Fig. 5(a), the intensity of clouds is calculated by using the octree. In Fig. 5(b), the octree is not used. As shown in these images, their qualities are almost the same. The sizes of the images are 720x480 and we use a desktop PC (PentiumIII 733MHz) with NVIDIA GeForce2GTS as the graphics hardware device. The computation times for Figs. 5(a) and (b) are 8 and 400 seconds, respectively. This result shows that the proposed algorithm can create images 50 times faster without losing the image quality.

Fig. 6 (see color plate) shows images of lightning under different conditions. Figs. 6(a) and (b) show multiple lightning strokes. The clouds and the glow are more brightened than those of Fig. 5. In Figs. 6(c) and (d), the color of lightning is set to pink. Figs. 6(e) and (f) show lightning at sunset. These images depict beautiful color variations. The rendering times for these images are from 10 to 25 seconds.

Next, the proposed method is applied to the flight simulation. We created an animation that had an airplane make a takeoff under thunderclouds. The initial points of lightning are determined randomly in clouds. Center positions of cloud voxels are selected for the initial points. The periods from occurrence to the extinction of lightning are also determined randomly. The periods are less than 0.5 seconds. The terrain is brightened in proportion to the intensity of lightning. Fig. 7 (see color plate) shows stills from the animation. In Figs. 7(a) and (b), lightning is not visible since it flashes in the clouds. Figs. 7(c) and (d) show the images of clouds just after the takeoff. Figs. 7(e) and (f) are the images close to the clouds.

Examples shown in this section demonstrate that the proposed method realizes the efficient generation of photo-realistic images including lightning.

6. Conclusion

An efficient method for rendering scenes including lightning has been proposed in this paper. Photo-realistic images can be generated by taking into account scattering effects due to clouds and atmospheric particles illuminated by lightning. The proposed method has the following advantages.

1. The atmospheric effects are efficiently rendered by preparing a lookup table. This stores the integrated intensity of the light scattering due to atmospheric particles.
2. The density distribution of the clouds is represented by using metaballs in the hierarchical structure using an octree. Using this structure, clouds illuminated by lightning are efficiently rendered by applying the idea of LOD to clouds.
3. Images of clouds viewed from the viewpoint are efficiently created by using hierarchical imposters.

There are several future projects that remain to be addressed. Firstly, the cloud-rendering algorithm using imposters may cause a ‘popping’ problem when the tolerance is greater than a certain value. That is, a sudden change in the clouds may be perceptible due to updating the imposter textures. A simple solution is to use smaller values for the tolerance [7]. However this results in increasing the rendering time. One method to address this problem has to be developed. Next, our method requires

about ten seconds to create a single image. Further acceleration of the process is needed for real-time simulations. Enhancing the reality is also an important issue. This may be achieved by taking into account the physics of lightning. For example, the intensity and color of lightning should be calculated by taking into account the energy of lightning, atmospheric condition, and so on. Moreover, for users that want to synthesize scenes including lightning, it would be useful to incorporate a function to specify the initial and destination points of lightning. Finally, since the intensity of the flash of lightning is very strong, methods for rendering high-dynamic range images may be incorporated [22].

Acknowledgments

This work is supported by Mitsubishi Electric Corporation.

References

[1] T. Reed, B. Wyvill, "Visual Simulation of Lightning," *Proc. SIGGRAPH'94*, pp. 359-364 (1994).

[2] P. Kruszewski, "A Probabilistic Technique for the Synthetic Imaginary of Lightning," *Computers & Graphics,* Vol. 23, No. 2, pp. 287-293 (1999).

[3] B. Sosorbaram, T. Fujimoto, K. Muraoka, N. Chiba, "Visual Simulation of Lightning Taking into Account Cloud Growth," *Proc. CG International 2001,* pp. 89-95 (2001).

[4] J. Stam, "Stable Fluids," *Proc. SIGGRAPH'99*, pp. 121-128 (1999).

[5] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, T. Nishita, "A Simple, Efficient Method for Realistic Animation of Clouds," *Proc. SIGGRAPH2000*, pp. 19-28 (2000).

[6] Y. Dobashi, T. Yamamoto, T. Nishita, "Interactive Rendering Method for Displaying Shafts of Light," *Proc. Pacific Graphics 2000*, pp. 31-37 (2000).

[7] M. J. Harris, A. Lastra, "Real-Time Cloud Rendering," *Computer Graphics Forum* (*Proc. EUROGRAPHICS2001*), to appear (2001).

[8] J. T. Kajiya, B. P. V. Herzen, "Ray Tracing Volume Densities," *Computer Graphics*, Vol. 18, pp. 165-174 (1984).

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

[10] R. V. Klassen, "Modeling the Effect of the Atmosphere on Light," *ACM Trans. on Graphics*, Vol.6, No.4, pp. 215-237 (1987).

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

[12] 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, pp. 293-302 (1987).

[13] 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, pp. 357-366 (1990).

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

[15] 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, pp. 471-482 (1998).

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

[17] G. Wyvill, A. Trotman, "Ray-Tracing Soft Objects," *Proc. CG International*, pp. 439-475 (1990).

[18] E. Paquette, P. Poulin, G. Drettakis, "A Light Hierarchy for Fast Rendering of Scenes with Many Lights," *Computer Graphics Forum* (*Proc. EUROGRAPHICS'98*), Vol. 17, No. 3, pp. 63-74 (1998).

[19] P. Maciel, P. Shirley, "Visual Navigation of Large Environments Using Textured Clusters," *Proc. 1995 symposium on Interactive 3D graphics*, pp. 95 (1995).

[20] G. Schaufler, "Dynamically Generated Imposters," *GI Workshop Modeling – Virtual Worlds – Distributed Graphics*, pp. 129-136 (1995).

[21] J. Shade, D. Lischinski, D. Salesin, T. Derose, J. Snyder, "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments," *Proc. SIGGRAPH'96*, pp. 75-82 (1996).

[22] P. E. Devebec, J. Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," *Proc. SIGGRAPH'97*, pp. 369-378 (1997).

(a) octree is used for clouds.

(b) octree is not used.

Figure 5. Examples of lightning.

Figure 6. Examples of lightning under different conditions. (a) and (b) show multiple lightning strokes. (c) and (d) show colored lightning (pink). (e) and (f) show lightning at sunset.

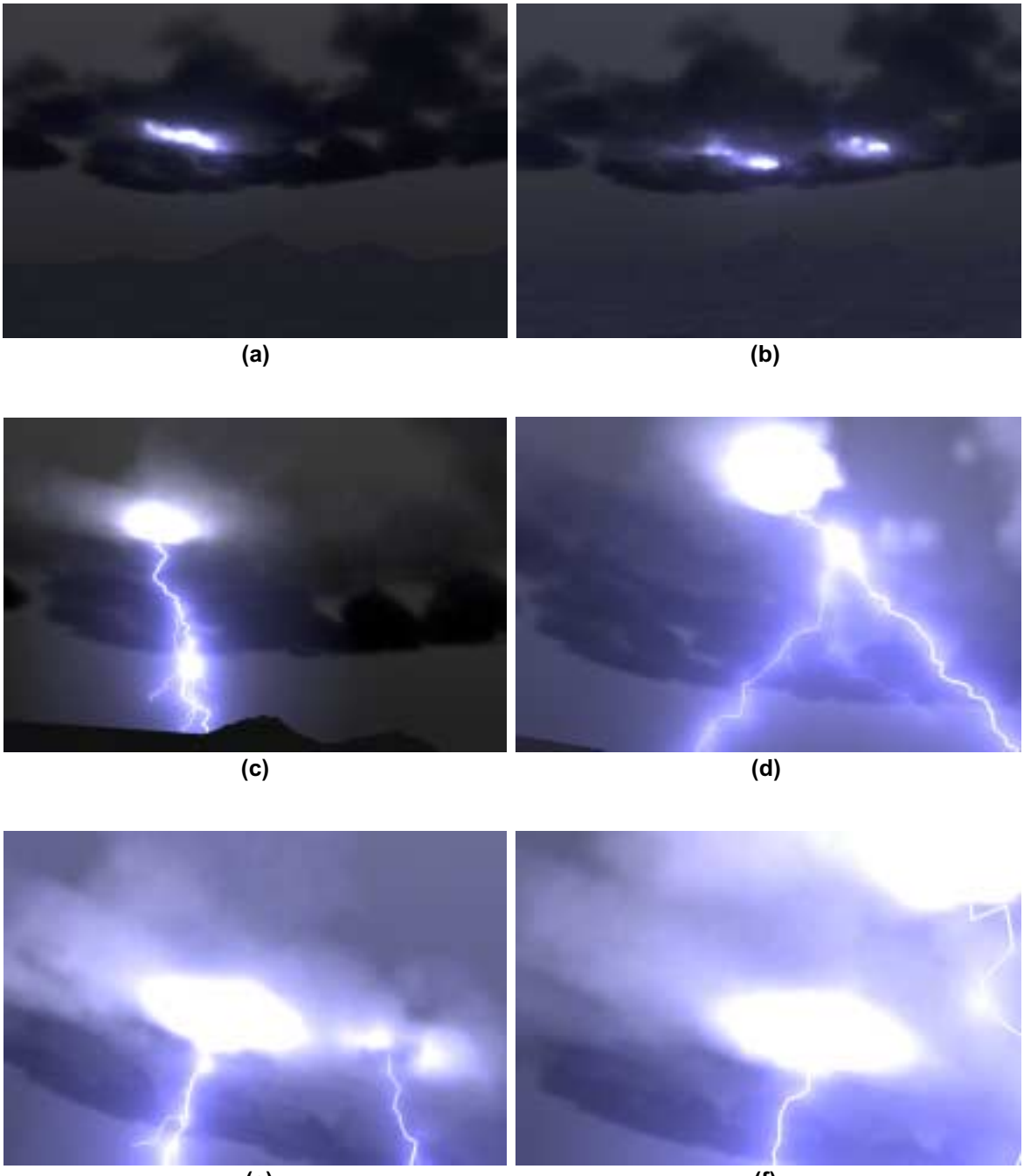


Figure 7. Application to flight simulation. (a) and (b) show flash of lightning in clouds. (c) and (d) show images just after takeoff. (e) and (f) show images close to clouds.