Modeling and Rendering of Various Natural Phenomena Consisting of Particles

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

Yoshinori Dobashi
Hokkaido University
Kita 31, Nishi 8, Kita-ku,
Sapporo, 060-8628, Japan
doba@nis-ei.eng.hokudai.ac.jp

Abstract

The simulation of various natural phenomena is one of the important research fields in computer graphics. In particular, aspects such as sky, clouds, water, fire, trees, smoke, terrains, desert scenes, snow and fog are indispensable for creating realistic images of natural scenes, flight simulators and so on. Therefore, a lot of researchers have been trying to develop methods for simulating and rendering these. In this paper, we focus on sky, clouds, smoke, desert scenes and atmospheric effects, such as shafts of light. These phenomena have the common feature that they are consist of the effects of small particles. To create realistic images, physical based simulation and rendering are required. In particular, the color greatly depends on the properties of light scattering due to particles. In general, however, the simulation and rendering of these images is assumed to be very time-consuming. This paper describes efficient methods for creating realistic images of such natural phenomena.

1. Introduction

In the field of computer graphics, various methods have been developed in order to display photo-realistic images by taking into account natural phenomena. Although there are many kinds of natural phenomena, many of them seem to have the common feature that they consist of the effects of small particles. Table 1 summarizes such natural phenomena. In these natural phenomena, light scattering/absorption due to particles is important. As shown in Table 1, depending on the size, Rayleigh and Mie scattering theories are applied to small and large particles respectively. The topics of this paper concern sky, clouds, smoke, desert scenes and atmospheric effects. These are very important elements in creating realistic images of natural scenes. The sky and clouds play an important role when making images for flight simulators or outdoor scenes. Methods of displaying smoke are used in various fields, such as visual simulation, entertainment, etc. We often find various beautiful ripple patterns that are made by wind on a sandy beach. Particles in the atmosphere produce the shafts of light caused by the headlights of automobiles, street lamps, studio spotlights, and light passing through stained glass windows, for example. As a result, there exists a great deal of research focusing on these topics.

Table 1 Natural phenomena related to particles

Material	Particles	Scattering
Sky (sky color, skylight)	Air molecules Aerosols	R M
Clouds	Water vapor, icicle	M
Smoke/gas	Water vapor, dust	M
Water/ liquid (color of water, shaft of light, caustic)	Water molecules	R
Fog / haze	Water particles	M
Atmosphere (shaft of light, volume light)	Water vapor	M
Snow	Snow flakes	M
Dunes	Sand particles	M
Saturn’s ring	icicle	M

R: Reyleigh scattering M: Mie scattering

There are two important issues for synthesizing photo-realistic images including the above natural phenomena. These are modeling and rendering.

Generally, the modeling process includes the creation of shapes of objects, their dynamics (motion/movement) and their physical properties such as surface reflectance. This is however not an easy task for objects such as clouds, smoke and sand dunes. They consist of small particles and therefore it is difficult to define definite their shapes. The simulation of their dynamics (movement) is also a difficult task, since their shape changes continuously with time. Therefore, a lot of modeling methods have been developed to address this problem. Using these methods, however, obtaining realistic-looking shapes and motion is very time consuming.

Rendering is the process of generating images by calculating colors for every pixel. The ray-tracing algorithm is often employed for generating images including the sky, clouds, smoke, desert scenes, and the atmospheric effects. Although the ray-tracing algorithm can create extremely realistic images, the computation time is very long. For example, the rendering of clouds and smoke requires the integration of the intensity of light scattered by small particles along the viewing ray. On the other hand, the processing speed of graphics hardware has become faster and faster recently. In addition, high performance graphics hardware is available even on low-end PCs. These facts have encouraged researchers to develop hardware-accelerated methods for rendering realistic images [1][2][3][4].

This paper introduces efficient methods for modeling and rendering of sky color, clouds, smoke, desert and other atmospheric effects. We use an image-based approach for modeling clouds viewed from space, the simulation of fluid dynamics for smoke, cellular automata for cloud motion and sand dunes. Efficient rendering is achieved by making use of graphics hardware. In the following sections the details of the methods are described with several examples.

2. The Sky

The color of the sky depends on the altitude of the sun, the atmospheric conditions and the viewing direction, changes that results in the variation of the appearance of buildings that reflect the sky color. The atmosphere consists of both air molecules and aerosols. The scattering due to the former obeys the Rayleigh scattering theory and the latter obeys the Mie scattering theory. As shown in Fig. 1, for Rayleigh scattering, the intensity of the sky, $L_\lambda(\mathbf{v})$, in the viewing direction, **v**, is calculated by the following equation [5].

$$L_\lambda(\mathbf{v}) = \frac{I_s(\lambda)kF(\alpha)}{\lambda^4}\int_0^{H_a} \rho(s)\exp(-t(s',\lambda)-t(s,\lambda))ds \text{ ,(1)}$$

where λ is the wave length, I_s the intensity of the sunlight, F the phase function of air molecules, H_a the distance between the viewpoint P_v and the top of the atmosphere P_a, ρ the density of air molecules, s' the distance between P and P_b, and s the distance between P and P_v (see Fig. 1).

Since the density ratio of air molecules varies exponentially with the height from the ground, the optical length, $t(s, \lambda)$, is calculated by the following equation.

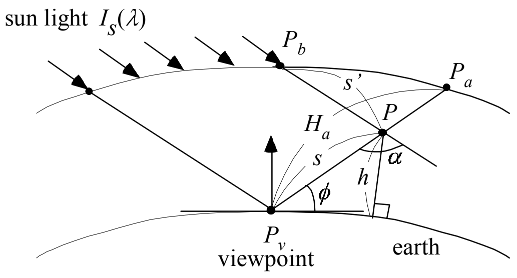

Figure 1: Calculation of sky color.

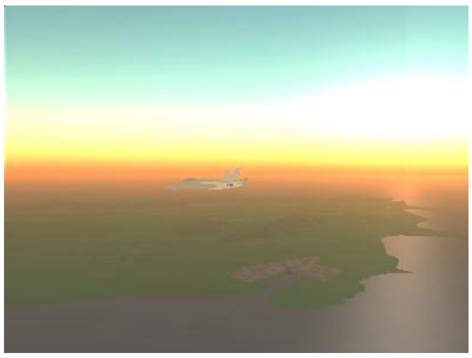

Figure 2: Example of sky color and fog effects.

$$t(s,\lambda) = \frac{4\pi k}{\lambda^4}\int_0^s \rho(l)dl = \frac{4\pi k}{\lambda^4}\int_0^s \exp(-\frac{h(l)}{H_0})dl \text{ ,} \qquad (2)$$

where H_0 is a constant called the scale height of the atmosphere and $h(l)$ is the height from the ground. Although both air molecules and aerosols must be taken into account in the actual physical phenomenon, only the scattering of air molecules is described here. For more details, see [5] [6]. To calculate the color of the sky from Eqs. (1) and (2), the intensity of the light arriving at the viewpoint, which is the sun light scattered and attenuated due to particles, must be integrated along the viewing ray. This requires the computation of the optical length of the path, P_bPP_v (see Fig. 1). The optical length is calculated by using Eq. (2). Since obtaining the analytical solution for the integral is difficult, the intensity of the sky must be calculated by the numerical integration method.

Since it takes time to calculate the color of the sky by using numerical integration every time the sky is displayed, the intensity distribution of the sky is calculated and stored in a look-up table. Then the sky is rendered in real-time as follows. The sky is considered to be a hemisphere with a large radius. The hemisphere is polygonized and the color of the sky at each vertex is obtained by using the look-up table. Then the hemisphere is display by using graphics hardware.

Fig. 2 shows an example of sky color and fog effects rendered by our method. In this figure, sky colors are calculated taking into account multiple scattering [7]. This realizes more realistic images of sky as shown in

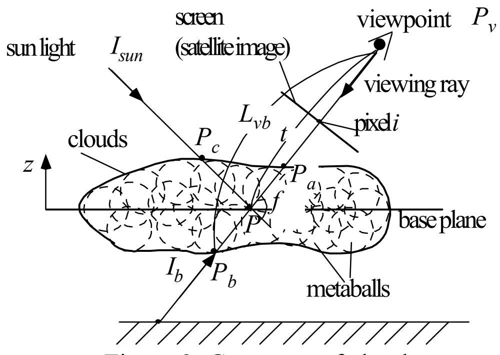

Figure 3: Geometry of clouds.

the figure.

3. Clouds

In this section, two methods for modeling clouds are described. The first one is for large-scale clouds such as typhoon viewed from space [8]. The second one is for motion of clouds viewed from ground [9]. Then, a hardware-accelerated method for rendering of clouds is explained [9].

3.1 Image-based Modeling of Clouds

For generating images of the earth viewed from outer space, large scale clouds such as typhoon have to be modeled. Our method uses an image-based approach for modeling such kinds of clouds. Our method creates three-dimensional clouds based on a satellite image [8]. As shown in Fig. 3, clouds are represented by a set of meatballs [10]. Parameters of metaballs (center positions, effective radii, and density values) are determined automatically so that a synthesized image of clouds modeled by metaballs coincides with the satellite image.

A density value at a point P in the cloud is given by the following equation.

$$\rho(P) \approx \sum_{j=1}^{N} q_j f(r_j, R_j)\,, \tag{3}$$

where N is the number of metaballs, q_j the density at the center of a metaball j, f the field function, and R_j the radius of the metaball. r_j is the distance between point P and the center position of a metaball, C_j, that is, $r_j = \left|P - C_j\right|$. We use the field function proposed by Wyvill et al. [10].

Determining parameters of the metaballs is equivalent to solving an inverse problem of determining the density distribution inside the clouds so that an image of synthesized clouds is similar to a satellite image. The problem is, however, very complicated and hard to solve. Therefore, we assume that the multiple scattering can be neglected. Furthermore, for modeling clouds, the attenuation of light due to cloud particles is approximated as a constant. Despite these assumptions, there is no unique solution to the problem. Therefore, the parameters of the metaballs are heuristically determined. First, each pixel of the satellite image is classified into either cloud region or background region. To do this, the satellite image is converted to a monotone one. Then pixels with intensities higher than a specified threshold are identified as clouds. Next, one metaball is added at the pixel with the maximum intensity in the cloud region. After that, its radius and density at the center are optimized and the approximated image is calculated using the clouds modeled by metaballs. Then a new metaball is added if the error between the satellite image and the approximated image is greater than a specified threshold. These processes are repeated until the error is less than the threshold.

Figure 4 shows an example of cloud created by our method. 3D Clouds in three-dimension are generated by using a two-dimensional infrared image of a typhoon taken from the meteorological satellite, “HIMAWARI”.

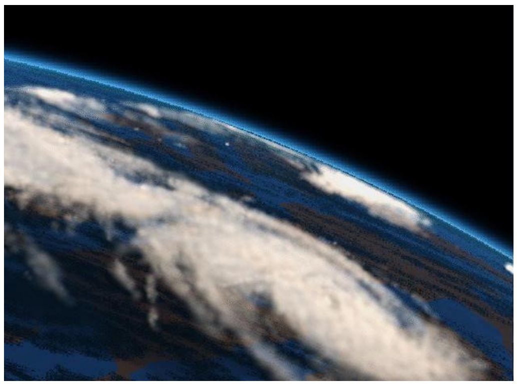

Figure 4: A typhoon viewed from space.

3.2 Simulation of Cloud Motion

The exact simulation of cloud motion is complex and computationally expensive. Therefore, we have developed a simple and efficient method [9]. In our method, the cloud motion is simulated using cellular automaton. The method can simulate cloud formation by simple transition rules. 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. *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. Cloud evolution is simulated by applying simple transition rules at each time step. The transition rules represent

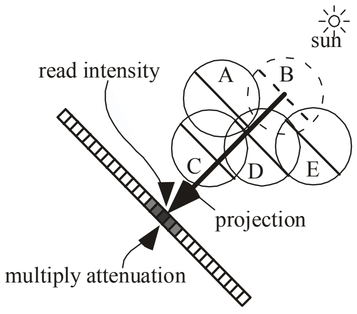

Figure 5: Algorithm for calculating the intensity of light reaching the center of metaballs.

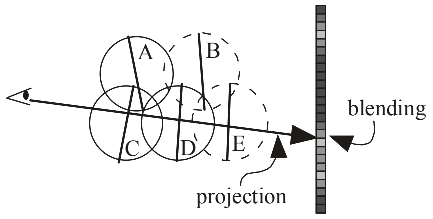

Figure 6: Algorithm for generating images.

formation, extinction, and advection by winds. For the cloud formation, Nagel et al. proposed the following three transition rules [11]. One of the disadvantages of Nagel's method is that cloud extinction never occurs since *cld*, after it has become 1, remains 1 forever. Therefore, our method simulates the cloud extinction by randomly changing *cld* to zero. 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, *hum* and *act*, are randomly set to 1. Moreover, to include the wind effect, all the variables are simply shifted toward the wind direction. These rules can realize the complex motion of clouds.

3.3 Efficient Rendering of Clouds Using Graphics Hardware

Rendering of clouds is based on the splatting algorithm using billboards [9]. The basic idea for applying it to cloud display is described here.

The color of clouds is calculated as follows. 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. Calculation of cloud color using splatting is as follows. First, textures for billboards are precalculated. Each element of the texture stores the attenuation ratio and cumulative density of the light passing through the metaball. 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

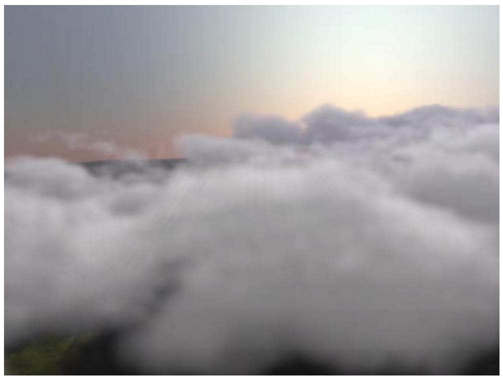

Figure 7: Simulation of sea of clouds.

metaball. 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. Next, attenuation ratio between the center of each metaball and the sun is calculated. 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. 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.

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, 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. 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 [12]). 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.

Figure 7 is an image of sea of clouds. This image was calculated on a desktop PC (PentiumIII 733MHz) with

NVIDIA GeForce2GTS and the image size is 640x480. The computation time was about 8 seconds. The proposed method realizes fast generation of realistic images.

4. Smoke

The problem of modelling smoke is it's complex behaviour from it's interactions with the surrounding air and obstacles is a topic of interest. We simulate these phenomena by combining the fluid dynamics simulation and the particle systems [13].

To model the density distribution of the smoke with less data, we consider a 'set' of smoke particles. We can deal with smoke as a combination of smoke clusters. This cluster is called a "puff." We use the metaball to define the density distribution of the puff. The motion of the smoke is simulated by moving meatballs according to the velocity field. This velocity field is calculated in advance by the fluid dynamics simulation.

A fluid is represented as a combination of a temperature field and velocity field. The rotational, buoyancy and convective components of the smoke motion are often modeled by the Navier-Stokes equations [14]. However, they don't consider the random velocity components of the turbulence. The random components should be taken into account for modeling the complex motion of the smoke. Therefore, we use the Reynolds equations, which consider the random velocity components of the flow [14].

The Reynolds equations fully describe the forces acting within the smoke flow, and consider the effects of random velocity components. The Reynolds equations are:

$$\frac{\partial \bar{u}}{\partial t} = -\left(\frac{\partial \bar{u}^2}{\partial x} + \frac{\partial \overline{uv}}{\partial y} + \frac{\partial \overline{uw}}{\partial z}\right) + \nu\left(\frac{\partial^2 \bar{u}}{\partial x^2} + \frac{\partial^2 \bar{u}}{\partial y^2} + \frac{\partial^2 \bar{u}}{\partial z^2}\right)$$
$$-\nu_t\left(\frac{\partial \overline{u'^2}}{\partial x} + \frac{\partial \overline{u'v'}}{\partial y} + \frac{\partial \overline{u'w'}}{\partial z}\right) - \frac{1}{\rho}\frac{\partial p}{\partial x},$$
$$\frac{\partial \bar{v}}{\partial t} = -\left(\frac{\partial \bar{v}^2}{\partial y} + \frac{\partial \overline{vu}}{\partial x} + \frac{\partial \overline{vw}}{\partial z}\right) + \nu\left(\frac{\partial^2 \bar{v}}{\partial x^2} + \frac{\partial^2 \bar{v}}{\partial y^2} + \frac{\partial^2 \bar{v}}{\partial z^2}\right)$$
$$-\nu_t\left(\frac{\partial \overline{v'^2}}{\partial y} + \frac{\partial \overline{v'u'}}{\partial x} + \frac{\partial \overline{v'w'}}{\partial z}\right) - \frac{1}{\rho}\frac{\partial p}{\partial y},$$
$$\frac{\partial \bar{w}}{\partial t} = -\left(\frac{\partial \bar{w}^2}{\partial z} + \frac{\partial \overline{wu}}{\partial y} + \frac{\partial \overline{wv}}{\partial z}\right) + \nu\left(\frac{\partial^2 \bar{w}}{\partial x^2} + \frac{\partial^2 \bar{w}}{\partial y^2} + \frac{\partial^2 \bar{w}}{\partial z^2}\right)$$
$$-\nu_t\left(\frac{\partial \overline{w'^2}}{\partial z} + \frac{\partial \overline{w'u'}}{\partial x} + \frac{\partial \overline{w'v'}}{\partial y}\right) - \frac{1}{\rho}\frac{\partial p}{\partial z} + F_{bv}, \qquad (4)$$

where ρ is the density, p the pressure, ν the coefficient of the viscosity, and ν_t the coefficient of eddy viscosity. In the right sides of Eq. 4, the third terms are the terms incorporating the random velocity components. In Eq. 4, there is a buoyancy term F_{bv} only used by the third equation. To calculate it, we have to consider the temperature field. Temperature T is defined as the sum of the average component $\bar{T}$ and the random component T'. To calculate the temperature field, we solve the equations of heat flow, which have some analogy with the Reynolds equations:

$$\frac{\partial \bar{T}}{\partial t} = -\left(\bar{u}\frac{\partial \bar{T}}{\partial x} + \bar{v}\frac{\partial \bar{T}}{\partial y} + \bar{w}\frac{\partial \bar{T}}{\partial z}\right) + \alpha\left(\frac{\partial^2 \bar{T}}{\partial x^2} + \frac{\partial^2 \bar{T}}{\partial y^2} + \frac{\partial^2 \bar{T}}{\partial z^2}\right) \qquad (5)$$
$$+\alpha_t\left(\frac{\partial^2 T'}{\partial x^2} + \frac{\partial^2 T'}{\partial y^2} + \frac{\partial^2 T'}{\partial z^2}\right),$$

where α are the coefficient of the heat diffusivity, and α_t is the coefficient of eddy and heat diffusivity.

To calculate above equations, we subdivide the whole space that contains the smoke flow into voxels as shown in Fig. 8. Within each voxel (i, j, k), there are some physical variables of the smoke, that is, the velocity vector ($u_{i,j,k}$, $v_{i,j,k}$, $w_{i,j,k}$), the temperature $T_{i,j,k}$, and the pressure $p_{i,j,k}$. At the center of each face of the voxel, we define a velocity component perpendicular to the face, and at the center of the voxel, we define variables of temperature and pressure. The velocity field is calculated numerically by using this representation.

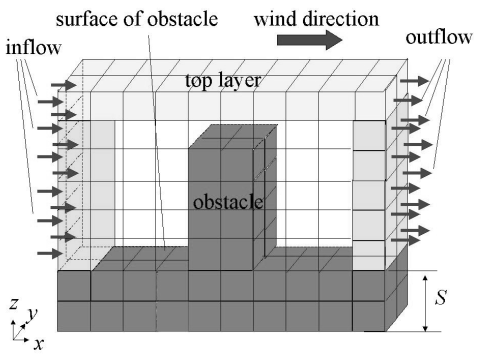

Figure 8: Voxelization of the simulation space.

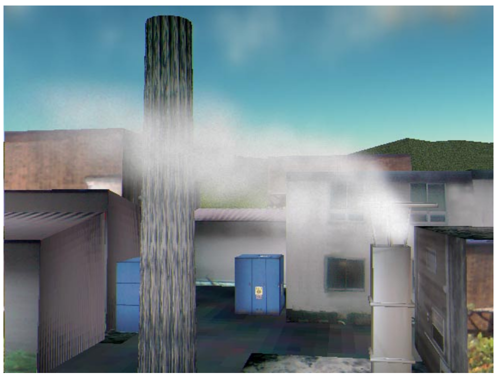
Figure 9: Smoke from a chimney.

Once the velocity field is obtained, the motion of smoke is simulated by moving meatballs. Basic idea of calculating colors of smoke is almost the same as that of clouds. Therefore, the method described in section 3.3 can be used for creating images of the smoke [13] [9].

Figure 9 shows an image of smoke.

5. Desert Scenes

In this section, methods for synthesizing realistic

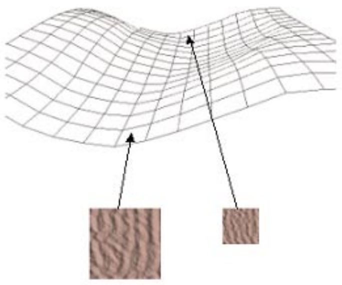
Figure 10: Bump-mapping using LODs.

images of desert scenes are described. First, our modeling method for images of sand dunes is briefly introduced. Then an efficient rendering method using the LOD principle is explained [15].

5.1 Modeling of Sand Dunes

A desert terrain includes sand dunes and wind-induced ripple patterns formed on the sand surface. The formation dynamics of wind-ripples and dunes have been investigated for a long time. We use Nishimori's model for modeling wind-ripples and dunes [16]. In this model, the wind-ripples and dunes are represented as height fields. The wind ripples are formed by calculating the saltation and creep of individual sand grains. The sand dunes are formed by considering the inclination in the movement of sand grains. The wind-ripples are formed when the wind force is within a critical range. When we see desert scenes, we often find wind-ripples formed on the dune surfaces. Since the scales of the wind-ripples and the dunes are completely different, they are simulated separately. Then the wind-ripples are mapped on the dunes as bump textures.

We use the formation dynamics of dunes and wind-ripples. The dynamics of sand grains consist of two elementary processes: saltation and surface creep. The saltation means the jump of sand grains caused by the wind. When a strong wind causes a sand grain rolling on the sand surface to collide with an obstacle, the grain is projected into the air, accelerated by the wind and then collides into other sand grains on the lee side. This jumping process is called saltation. Sand grains may move along the sand surface without jumping up. This movement is called creep. These processes are calculated using the cellular automata method. To do this, two-dimensional array is prepared. Each element of the array stores the height of the sand dunes. Then, a height field is generated randomly and stored in the array. Applying simple transition rules for each element of the array gradually generates the sand dunes/ripple patterns. For more detail, see [15].

5.2 Rendering of Desert Scenes

As shown in Fig. 10, a mesh (set of quadrilaterals) is

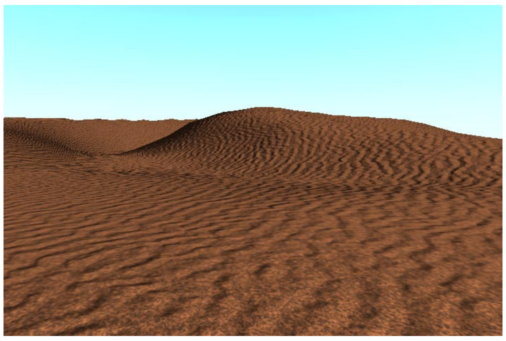
Figure 11: A desert scene.

generated by using the height field stored in the array described in the previous section. The desert scenes are rendered by using these quadrilaterals. The outline of the rendering process is as follows.

1. Calculate wind-ripple's normal vectors.
2. For each quadrilateral *F* of the height field mesh of the dune:
   (a) Backface culling and the view frustum culling is performed.
   (b) Calculate LOD by taking the distance from the viewpoint to *F*.
   (c) Calculate the wind-ripples texture. The texture resolution is determined by LOD and is mapped onto *F*.

To represent the wind-ripple patters on the dune surface, we generate the textures using the bump mapping technique [17] by using the normal vectors that are calculated from the wind-ripples height field. This method allows us to render a realistic image without increasing to a large computation time, since each quadrilateral has a different texture. To solve this problem, we use the LOD technique. The basic idea of the LOD is to use simpler versions of an object as it gets farther from the viewer. For the quadrilaterals far from the viewpoint, the texture is insignificant because the quadrilateral takes only a few pixels on the screen. Therefore, we use textures of different resolutions according to the Olds. Our method is as follows. First, we calculate the normal vectors of wind-ripples. Using these vectors, we prepare the normal vectors for different resolutions. For example, when the original size of wind-ripple's height field is 256×256, we pre-calculate the normals of the 256×256, 128×128, 64×64, and 32×32 resolutions.

Finally, we calculate the textures for each quadrilateral of the dune using the resolution determined by the quadrilateral's LOD. The LOD is calculated from the distance from the viewpoint to the quadrilateral. For example, as shown in Fig. 10, a large texture is mapped onto a quadrilateral near the viewpoint, and vice versa. This method resembles

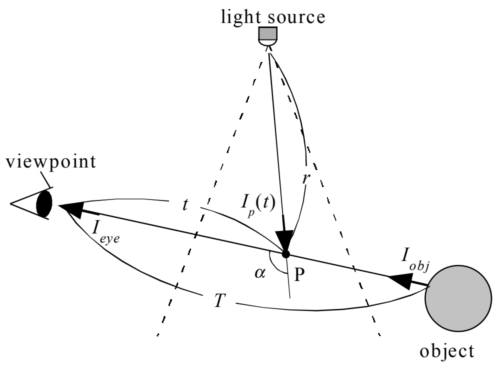

Figure 12: Shading model for atmospheric scattering.

mipmapping [18], but each texture is different. Therefore we cannot use mipmapping. Using this method, the rendering time is greatly reduced and the quality of the resulting images is almost the same as that of images without LODs.

Figure 11 shows an example of a desert scene. This image is calculated using a Pentium III 750 MHz with a NVIDIA GeForce256 video card and the image size is 720×480. The rendering time is 3.5 sec.

6. Atmospheric Effects

One of the important elements in creating realistic images is the effect of atmospheric scattering [19][20][21]. Examples of such effects include light beams due to spotlights and shafts of light from the sun's rays. This section introduces a method for displaying shafts of light at interactive rates by making use of the graphics hardware [22].

6.1 Shading Model for Atmospheric Effects

Fig. 12 shows the concept of the calculation for light scattering. In Fig. 12, a point light source is assumed. In general, the intensity of light reaching the viewpoint is expressed by the following equation.

$$I_{eye} = I_{obj}\beta(T) + \int_0^T F(\alpha)H(t)I_p(t)\beta(t)dt, \qquad (6)$$

where I_{eye} is the intensity reaching the viewpoint, I_{obj} is the intensity of an object, $\beta(t)$ the attenuation ratio due to atmospheric particles between the viewpoint and point P on the viewing ray, t the distance between the viewpoint and point P, T the distance between the viewpoint and the object, and $I_p(t)$ the intensity of light from the light source reaching point P. $H(t)$ is a visibility function that returns the value 1 if the light source is visible from point P, or 0 otherwise. $F(\alpha)$ is a phase function of the atmospheric particles and α is the phase angle (see Fig. 12). Under the assumption that the density of the particles is uniform, the attenuation term $\beta(t)$ and $I_p(t)$ is calculated analytically.

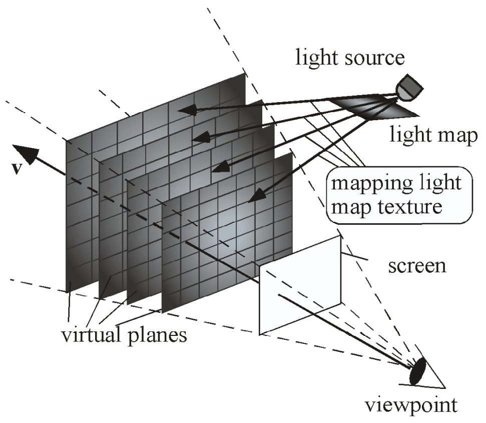

Figure 13: Basic concept of rendering shafts of light.

In the above equation, the second term indicates the total intensity of scattered light due to atmospheric particles and is directly related to the shafts of light. Most of the graphics hardware can calculate the first term by using a function for simulating fog effects. The next section explains our method for calculating the second term by making use of graphics hardware.

6.2 Hardware-accelerated Rendering of Shafts of Light

The intensity of light scattered from particles has to be integrated along the viewing ray to display the shafts of light. Fig. 13 shows the basic idea of our method. Virtual planes are placed in front of the viewpoint in order to integrate the scattered light. Each virtual plane is parallel to the screen and represented by a lattice mesh (see Fig. 13). The idea for calculating the total light reaching the viewpoint is as follows. First, the intensity of the light scattered at each lattice point is calculated and then stored. The total intensity is the sum of the intensities of all the virtual planes. It is computed by rendering the virtual planes with an additive blending function. The luminous intensity distribution of the light source can be taken into account by mapping a light map texture [23] onto the virtual planes with a multiplicative mapping function (see Fig. 13).

Shadows cast on particles in the atmosphere are very important; when there are objects in an illuminated volume, non-illuminated parts arise within it. To display the shadow in the atmosphere, non-illuminated parts of the virtual planes must be detected. To achieve this, we have implemented the idea of the shadow map [23] by creating shadow textures for each virtual plane. That is, before rendering each virtual plane, an image of the shadow cast on it is created. The image is then used to mask the non-illuminated parts of the virtual plane by mapping it as a texture with a multiplicative blending function.

Figs. 14 and 15 are examples of rendering shafts of light. Fig. 14 shows shafts of light caused by sunlight passing through stained glass windows. Fig. 15 shows shafts of light caused by spotlights in a opera house. The computation was done on a desktop PC (PentiumIII 733MHz) with NIVIDIA GeForce256. The sizes of images are 640x480. Computation times for Fig. 14 and 15 are 0.7 and 2.1 seconds, respectively.

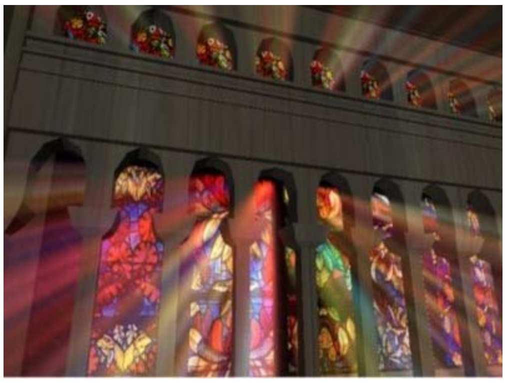
Figure 14: Shafts of light caused by sunlight through stained glass windows.

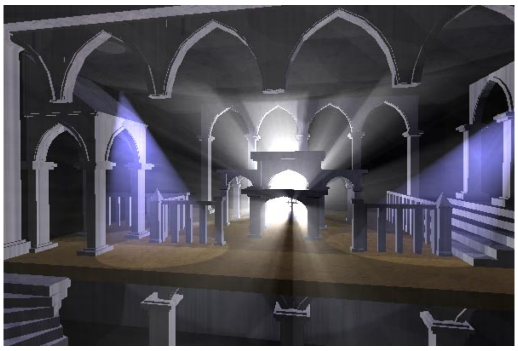
Figure15: Shafts of light caused by spotlights.

7. Conclusion

This paper has discussed efficient algorithms for synthesizing realistic images of various natural phenomena consisting of particles. In this paper, methods for modeling clouds, smoke, and sand dunes have been introduced. Realistic shapes and motion can be generated by using these methods. We have also described methods for rendering sky, clouds, smoke, sand dunes, and shafts of light. The scattering/absorption due to particles have to be taken into account to create images of these. The graphics hardware was utilized to accelerate the calculation. As shown in the examples, the methods discussed here can give us photo-realistic images.

Acknowledgment

The authors would like to acknowledge Miss Keiko Sotome (Joshibi Univ. of Art and Design) for her help in making the geometric data for the church and the opera house in Figs. 14 and 15.

References

[1] E. Ofek, A. Rappoport, "Interactive Reflections on Curved Objects," Proc. of SIGGRAPH'98, 1998, pp. 333-342.

[2] W. Heidrich, H. P. Seidel, "Realistic, Hardware-Accelerated Shading and Lighting," Proc. of SIGGRAPH'99, 1999, pp. 171-178.

[3] J. Stam, "Stable Fluids," Proc. of SIGGRAPH'99, 1999, pp. 121-128.

[4] B. Cabral, M. Olano, P. Nemec, "Reflection Space Image Based Rendering," Proc. of SIGGRAPH'99, 1999, pp. 165-170.

[5] 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.

[6] 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.

[7] T. Nishita, Y. Dobashi, K. Kaneda, H. Yamashita, "Display Method of the Sky Color Taking into Account Multiple Scattering," Proc. Pacific Graphics'96, 1996, pp. 117-132.

[8] 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.

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

[10] G. Wyvill and A. Trotman, "Ray-Tracing Soft Objects," Proc. of CG International, 1990, pp.439-475.

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

[12] L. Westover, "Footprint Evaluation for Volume Rendering," Computer Graphics, Vol. 24, No. 4, 1990, pp. 367-376.

[13] S. Yoshida, T. Nishita, "Modeling of Smoke Flow Taking Obstacles into Account," Proc. Pacific Graphics 2000, 2000, pp. 135-144.

[14] G.K.Batchelor, "An Introduction to Fluid Dynamics," Cambridge At The University Press, 1967.

[15] K. Onoue, T. Nishita, "A Method for Modeling and Rendering Dunes with Wind-Ripples,", Proc. Pacific Graphics 2000, 2000, pp. 427-428.

[16] H. Nishimori and N. Ouchi, "Formation of Ripple Patterns and Dunes by Wind-Blown Sand," Physical Review Letters Vol.71, No.1, 1993, pp.197-200.

[17] J.Blinn, "Simulation of wrinkled surfaces," Computer Graphics, Vol.12, No.3, 1978, pp.286-292.

[18] L.Williams, "Pyramidial Parametrics," Computer Graphics, Vol.17, No.3, 1983, pp.1-11.

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

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

[21] 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.

[22] Y. Dobashi, T. Yamamoto, T. Nishita, "Interactive Rendering Method for Displaying Shafts of Light," Proc. Pacific Graphics2000, 2000, pp. 31-37.

[23] M. Segal, C. Korobkin, R. V. Widenfelt, J. Foran, P. E. Haeberli, "Fast Shadows and Lighting Effects Using Texture Mapping," Computer Graphics, Vol. 26, No. 2, 1992, pp. 249-252.