Pacific Graphics 2010
P. Alliez, K. Bala, and K. Zhou
(Guest Editors)

Volume 29 (2010), Number 7

Modeling of Clouds from a Single Photograph

Yoshinori Dobashi Yusuke Shinzo Tsuyoshi Yamamoto

Hokkaido University, Sapporo, Japan

Abstract

In this paper, we propose a simple method for modeling clouds from a single photograph. Our method can synthesize three types of clouds: cirrus, altocumulus, and cumulus. We use three different representations for each type of cloud: two-dimensional texture for cirrus, implicit functions (metaballs) for altocumulus, and volume data for cumulus. Our method initially computes the intensity and the opacity of clouds for each pixel from an input photograph, stored as a cloud image. For cirrus, the cloud image is the output two-dimensional texture. For each of the other two types of cloud, three-dimensional density distributions are generated by referring to the cloud image. Since the method is very simple, the computational cost is low. Our method can generate, within several seconds, realistic clouds that are similar to those in the photograph.

Categories and Subject Descriptors (according to ACM CCS): I.3.m [Computer Graphics]: Miscellaneous—

1. Introduction

The realistic display of clouds is one of the important research topics in computer graphics. Clouds play an important role in creating synthetic images of outdoor scenes. In order to display realistic clouds, the density distribution of clouds requires to be defined. Many methods have been developed for this purpose.

There are two major approaches to modeling clouds: the procedural approach and the physically based approach. The procedural approach can synthesize realistic clouds with a low computational cost [EMP*02] . However, in order to create the desired types of cloud, many parameters have to be determined manually by trial and error processes. The physically based approach, on the other hand, generates clouds by simulating the physical process of cloud formation [DKNY08]. However, the computational cost is much more expensive than the procedural approach.

In this paper, we propose an alternative method for the modeling of clouds. Our method uses a single photograph to synthesize density distribution of clouds. The goal of our method is not to reproduce exactly the same clouds as those in the photograph, as reconstructing three-dimensional clouds from a single photograph is an extremely difficult problem and is almost impossible. Instead, our method uses the photograph as a guide to synthesize clouds that look similar to those in the photograph. Our method can synthesize three types of cloud: cirrus, altocumulus, and cumulus (or cumulonimbus). Cirrus is generally thin and self-shadows are seldom observed. Therefore, cirrus is modeled as a two-dimensional texture. Altocumulus and cumulus possess volumetric features and three-dimensional density distributions must be generated. Since our method is very simple, the computational cost is low, and realistic clouds can be synthesized within several seconds.

2. Related Work

This section briefly discusses previous methods for modeling clouds. For more detailed discussion on cloud modeling and simulation, please refer to [Har03].

Procedural modeling is the most popular approach to modeling clouds and many methods have been proposed. Voss used the idea of fractals for modeling clouds [Vos83]. Gardner proposed a method using textured ellipsoids for visual simulation of clouds [Gar85]. Ebert et. al. developed a method combining metaballs and a noise function [Ebe97]. Sakas modeled clouds by using spectral synthesis [Sak93]. More recently, Schpok et al. have developed a real-time system for the procedural modeling of clouds [SSEH03]. A more detailed explanation of the procedural modeling of clouds can be found in [EMP*02]. These methods can gen-

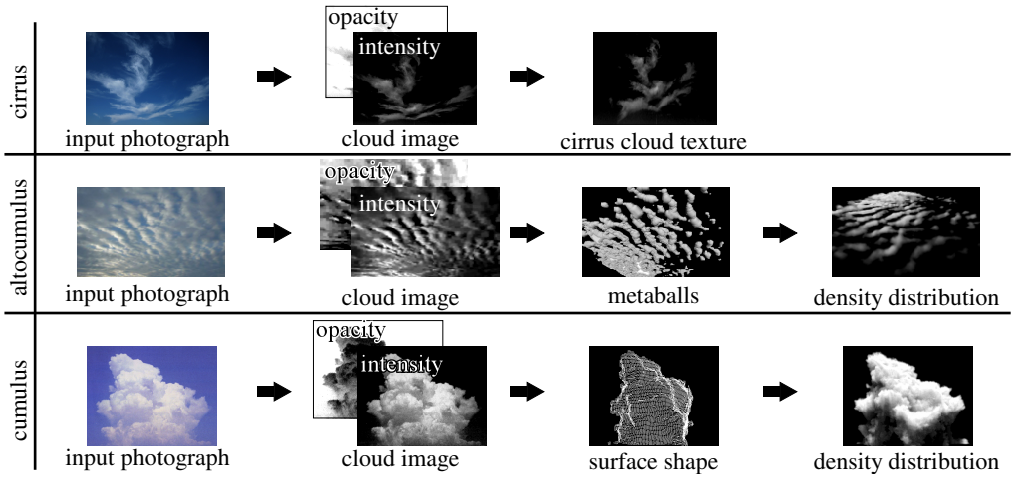


Figure 1: *Overview of our cloud modeling processes.*

erate realistic clouds, but many parameters are required to be specified by trial and error.

Clouds can be generated by physically based simulation of the cloud formation process. Kajiya and Herzen solved atmospheric fluid dynamics, numerically, for modeling cumulonimbus clouds [KH84]. Miyazaki et al. proposed a method for modeling various types of cloud by using the method called a coupled-map lattice, being an extended version of cellular automata [MYND01]. They also proposed a method for simulating the cloud formation process by improving the method proposed by Kajiya and Herzen [KH84]. More recently, Dobashi et al. proposed a method for controlling the simulation to generate desired shapes of clouds [DKNY08], although the method is limited to cumuliform clouds. By using these methods, realistic clouds can be created. However, one of the problems with these methods is that the computational cost is very high.

The most closely related method to ours is the one proposed by Dobashi et al [DNYO98]. This method uses infrared satellite images and metaballs for modeling earth-scale clouds. The background of the clouds in satellite images is very dark and therefore the effect of the background intensity on the cloud intensity is negligible. However, for the photograph taken from the ground, we need to remove the effects of the background sky. Our method initially estimates the background image of the sky for this purpose. The similarity between our method and [DNYO98] is in the use of metaballs. However, the previous method determines all the parameters of the metaballs (center position, center density, and radius) by an exhaustive search algorithm, resulting in a large computational cost. In our method, only center densities are determined by optimization, reducing the computational cost. In addition, [DNYO98] pays no attention to the types of cloud, while we develop different methods according to the cloud types.

For altocumulus and cumulus, we employ a geometry based approach; the surface shapes of clouds are determined from an input photograph. The density distributions within the shapes are then generated. Although geometry based methods for modeling clouds have been proposed (e.g., [BN02] [NND96]), their purpose is not to generate the clouds from a photograph. Therefore, it would be difficult to generate clouds that are similar to those in the photograph. Our method is more tightly coupled to the photograph.

3. Overview of Our Method

Fig. 1 shows an overview of our method. As mentioned before, our method can generate three types of cloud: cirrus, altocumulus, and cumulus, as shown in the figure. Although we propose three methods for modeling these clouds, there is a common process in all of the three methods, i.e., calculation of the cloud image. The cloud image is calculated from the input photograph, and an intensity and an opacity of the clouds are stored at each pixel. To calculate the cloud image, our method initially creates the sky image by removing clouds from the photograph. This is achieved by estimating the sky colors behind the clouds. Then, the intensity and the opacity are calculated by comparing the input photograph with the sky image. After creating the cloud image, one of the three methods is carried out according to the type of the clouds in the photograph. The cloud type is specified by the user. We assume that the camera parameters (the camera position and viewing angle) are also provided by the user.

Cirrus clouds are very thin and self-shadows are seldom observed. Therefore, we model cirrus as a two dimensional texture. The cloud image described above is used to create the cirrus texture (see the top column in Fig. 1).

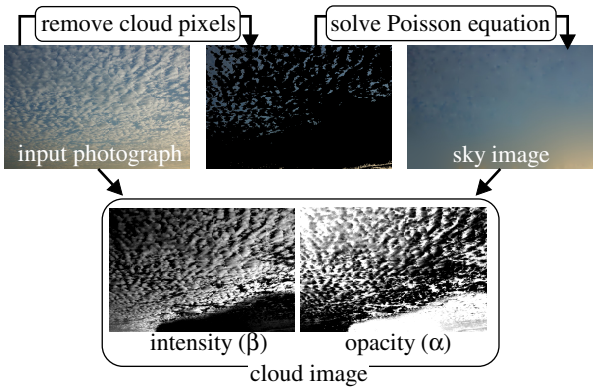


Figure 2: *Calculation of cloud image.*

Altocumulus is also thin, but self-shadows are observed. Therefore, a three-dimensional density distribution must be defined. We use metaballs to define the density distribution, as shown in the second column in Fig. 1. A metaball is a sphere, inside which a field function is assigned. We use the function proposed by Wyvill and Trotman [WT90]. Metaballs are often used for defining a surface of a soft object, but our method uses them to define the density distribution as a sum of the field functions. A metaball possesses three parameters: a center position, a radius, and a center density. Our method determines these parameters so that the cloud image is approximated by the metaballs.

Cumulus clouds possess depth. Our method initially generates a surface shape of the clouds by calculating the thickness at each pixel, as shown in the third row of Fig. 1. Our method computes the surface shape by assuming that the clouds are thicker in the center and the thinner at the boundary. The density inside the shape is then generated, employing a procedural approach.

4. Calculation of Cloud Image

Fig. 2 shows the procedure for calculating the cloud image. The first process is to create the sky image by estimating sky colors behind the clouds in the input image. Then, the intensity and the opacity of the clouds are calculated by comparing the input photograph with the sky image.

4.1. Calculation of Sky Image

To create the sky image, each pixel in the input image is roughly classified into either a cloud pixel or a sky pixel, and the cloud pixels are removed (see Fig. 2). The sky image is created by extending the colors of the surrounding sky pixels into the removed cloud areas. For the classification, we use a chroma of each pixel. Colors of clouds are generally white (or gray) and therefore the chroma of a cloud pixel is expected to be small. This fact allows us to identify the cloud pixels by comparing the chroma of each pixel with the user-specified threshold, ε_c. Let us denote the intensity of pixel p of the input photograph as $I(p,\lambda)(\lambda = R,G,B)$. Then, the chroma of pixel p, $S(p)$, is calculated from:

$$S(p) = (I_{max}(p) - I_{min}(p))/I_{max}(p), \tag{1}$$

where,

$$I_{max}(p) = \max_{\lambda=R,G,B} I(p,\lambda), \quad I_{min}(p) = \min_{\lambda=R,G,B} I(p,\lambda).$$

If $S(p)$ is smaller than ε_c, then pixel p is labeled as a cloud pixel.

Next, the sky image is created by interpolating the colors of the cloud pixels from the surrounding sky colors. For this interpolation, we borrow an idea from the image editing technique, called Poisson image editing [PGB03]. i.e., we solve the following Poisson equation for the cloud pixel p_c.

$$\Delta I_{sky}(p_c,\lambda) = 0, \tag{2}$$

where Δ indicates the Laplacian operator and $I_{sky}(p_c,\lambda)$ is intensity at pixel p_c in the sky image. The above equation is numerically solved for only the cloud pixels, p_c. In solving this equation, colors of the sky pixels neighboring the removed cloud pixels are used as boundary conditions. This method provides us with the smooth and seamless intensity distribution of the sky (see Fig. 2).

4.2. Calculation of Cloud Intensity and Opacity

Before explaining the details of the calculation of the cloud image, let us start with a discussion of the intensity of clouds.

Let us consider a situation shown in Fig. 3, where clouds are illuminated by the sun. The intensity of light reaching point $\mathbf{y}$ from the sun and scattered into direction ω is represented by:

$$\begin{aligned} I_1(\mathbf{y},\omega,\lambda) &= \rho(\mathbf{y})g(\mathbf{x}_s,\mathbf{y})F(\theta_1)I_{sun}(\lambda) \\ &= h_1(\mathbf{y},\omega)I_{sun}(\lambda), \\ h_1(\mathbf{y},\omega) &= \rho(\mathbf{y})g(\mathbf{x}_s,\mathbf{y})F(\theta_1), \end{aligned}$$

where λ is the wavelength, I_{sun} the sunlight intensity, ρ the density of clouds, F the phase function, θ_1 the phase angle between the sunlight direction and ω, and $g(\mathbf{x}_s,\mathbf{y})$ the attenuation ratio due to cloud particles between the sun $\mathbf{x}_s$ and point $\mathbf{y}$. Since the phase function of cloud particles is almost independent of the wavelength of light [NND96], h_1 is independent of λ. Then, the intensity of light for $k(>1)$th order scattering can be represented by the following recursive equations.

$$\begin{aligned} I_k(\mathbf{x},\omega,\lambda) &= \int_{V_c} \rho(\mathbf{x})g(\mathbf{y},\mathbf{x})F(\theta_k)G(\mathbf{y},\mathbf{x})I_{k-1}(\mathbf{y},\omega',\lambda)d\mathbf{y} \\ &= \int_{V_c} h_k(\mathbf{x},\omega)I_{k-1}(\mathbf{y},\omega',\lambda)d\mathbf{y}, \\ h_k(\mathbf{x},\omega) &= \rho(\mathbf{x})g(\mathbf{y},\mathbf{x})F(\theta_k)G(\mathbf{y},\mathbf{x}), \end{aligned}$$

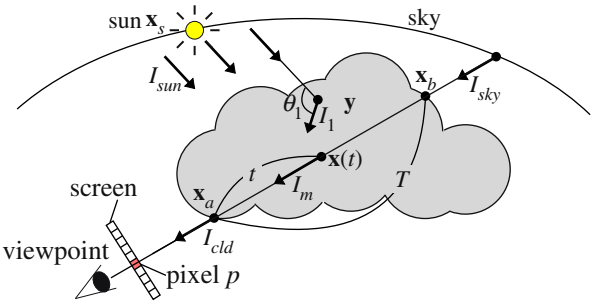


Figure 3: *Intensity of clouds.*

where ω' is the direction from $\mathbf{y}$ to $\mathbf{x}$, G the form factor calculated by the geometric relation between $\mathbf{x}$ and $\mathbf{y}$, and θ_k the angle between ω and ω'. V_c indicates all the points within the cloud volume. By expanding the recursion, I_k can be rewritten as:

$$\begin{aligned} I_k(\mathbf{x},\omega,\lambda) &= \int\cdots\int h_k h_{k-1}\cdots h_1 I_{sun}(\lambda) d\mathbf{y}_1 d\mathbf{y}_2\cdots d\mathbf{y}_{k-1}. \\ &= H_k(\mathbf{x},\omega) I_{sun}(\lambda), \\ H_k(\mathbf{x},\omega) &= \int\cdots\int h_k h_{k-1}\cdots h_1 d\mathbf{y}_1 d\mathbf{y}_2\cdots d\mathbf{y}_{k-1}, \end{aligned}$$

where $\mathbf{y}_1,\mathbf{y}_2,\cdots,\mathbf{y}_{k-1}$ are temporal variables for the integrals. Note that, for simplicity, we omit some symbols, such as V_c and ω', from the above equations. The total intensity of light at point $\mathbf{x}$ *after* multiple scatterings, I_m, is then expressed by:

$$I_m(\mathbf{x},\omega,\lambda) = \sum_{k=1}^{\infty} H_k(\mathbf{x},\omega) I_{sun}(\lambda) = H_{sun}(\mathbf{x},\omega) I_{sun}(\lambda), \quad (3)$$

where,

$$H_{sun}(\mathbf{x},\omega) = \sum_{k=1}^{\infty} H_k(\mathbf{x},\omega)$$

As shown in Eq. (3), I_m can be represented by the product of the sunlight intensity, I_{sun}, and light transfer function, H_{sun}, between the sun and point $\mathbf{x}$.

The intensity of light reaching the viewpoint is obtained by accumulating I_m along the viewing ray. The light of the sky behind the clouds also reaches the viewpoint after being attenuated by the cloud particles. Thus, the intensity of light reaching viewpoint through pixel p is expressed by:

$$\begin{aligned} I_{cld}(p,\lambda) &= \int_0^T \rho(\mathbf{x}(t)) g(\mathbf{x}(t),\mathbf{x}_a) I_m(\mathbf{x}(t),\omega_v,\lambda) dt \\ &+ I_{sky}(p,\lambda) g(\mathbf{x}_a,\mathbf{x}_b) \end{aligned} \quad (4)$$

where $\mathbf{x}_a$ and $\mathbf{x}_b$ are the intersection points between viewing ray and the clouds (see Fig. 3), T the thickness of the clouds, ω_v the direction of the viewpoint viewed from point $\mathbf{x}(t)$, I_{sky} the intensity of the sky behind the clouds. The first term on the right indicates the intensity of clouds, and the second term is the intensity of the light of the sky, attenuated by the cloud particles. Next, we split $I_{sun}(\lambda)$ into its intensity, i_{sun}, and color components, $c_{sun}(\lambda)$, i.e., $I_{sun}(\lambda) = i_{sun} c_{sun}(\lambda)$, and insert Eq. (3) into Eq. (4) to obtain the following equation.

$$I_{cld}(p,\lambda) = \beta(p) c_{sun}(\lambda) + \alpha(p) I_{sky}(p,\lambda), \quad (5)$$

where,

$$\alpha(p) = g(\mathbf{x}_a,\mathbf{x}_b), \quad (6)$$

$$\beta(p) = i_{sun} \int_0^T \rho(\mathbf{x}(t)) g(\mathbf{x}(t),\mathbf{x}_a) H_{sun}(\mathbf{x}(t),\omega_v) dt \quad (7)$$

α and β correspond to the opacity and the intensity of clouds, respectively.

Our method computes α and β for each cloud pixel, p_c, by assuming that $I_{cld}(p_c,\lambda)$ is equal to the pixel intensity of the input image, $I(p,\lambda)$. That is,

$$\begin{aligned} I(p_c,\lambda) &= \beta(p_c) c_{sun}(\lambda) + \alpha(p_c) I_{sky}(p_c,\lambda) \quad (8) \\ (\lambda &= R,G,B) \end{aligned}$$

In order to solve above equations in terms of α and β, we need to know c_{sun} and I_{sky}. For I_{sky}, we use the sky image, calculated by using the method described in Section 4.1. However, we still have five unknowns (α, β, $c_{sun}(R)$, $c_{sun}(G)$, $c_{sun}(B)$) for each pixel. Since c_{sun} should be the same for all the pixels, we can obtain α and β (and c_{sun}) by solving the equations for three neighboring pixels, simultaneously. However, c_{sun} obtained in this way is not necessarily the same for all the pixels, because of some noise involved in the input image. So, we employ the following two-step approach. First, for each pixel, our method calculates optimal sun color, $c_{sun}(\lambda)$, that minimizes the following function:

$$\sum_{\lambda=R,G,B} \left(I(p_c,\lambda) - \beta(p_c) c_{sun}(\lambda) - \alpha(p_c) I_{sky}(p_c,\lambda) \right)^2.$$

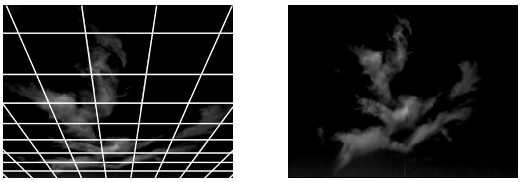
This minimization problem is solved by fully searching the RGB color space. Next, the average sun color of all pixels is calculated. By using the average sun color, α and β for each pixel is re-calculated. α and β for each pixel are then stored in the cloud image.

5. Modeling of Clouds

Using the cloud image obtained from the previous section, three types of cloud (cirrus, altocumulus, and cumulus) are generated. The following subsections describe the details of our method for each of the cloud types.

5.1. Cirrus

As mentioned before, cirrus clouds are represented as a two-dimensional texture using the cloud image. However, we need to remove the effect of the perspective transformation in the input photograph. In order to achieve this, a cloud plane, where the clouds are assumed to exist, is interactively



(a) cloud plane. (b) cirrus cloud texture.

Figure 4: *Creation of cirrus cloud texture.*

specified by the user (see Fig. 4). Then, by projecting the cloud image onto the cloud plane, the cirrus cloud texture is created.

5.2. Altocumulus

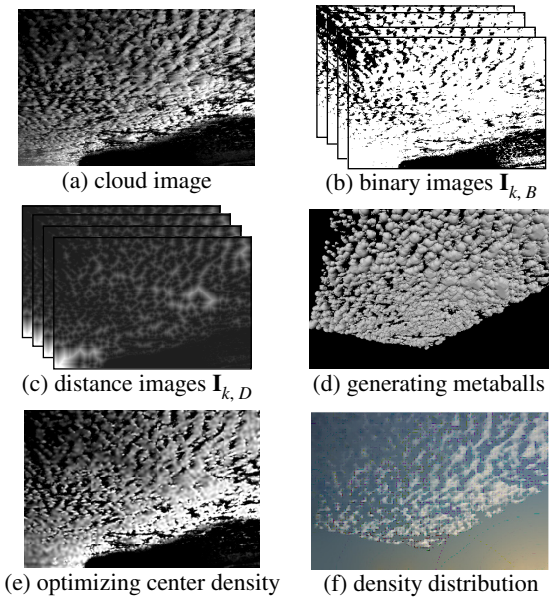
Fig. 5 shows the procedure for modeling altocumulus. Altocumulus clouds typically consist of many small cloud cells, as shown in Fig. 5(a). The basic idea is to generate a set of metaballs to approximate the cloud image. Our method generates metaballs under the following assumptions. (1) The clouds look unnatural if the thicknesses become larger than the size of the cloud cells and (2) the thickness should be different if the opacity (α) is different. The details of the metaball generation process are described in the following.

First, a set of binary images $\mathbf{I}_{k,B}(k = 1,2,\cdots,n)$ are created by using a set of thresholds for α (Fig. 5(b)). That is, a white color is stored at pixel p of $\mathbf{I}_{k,B}$ if $k\Delta\alpha - \Delta\alpha/2 < \alpha(p) \leq (k+1)\Delta\alpha + \Delta\alpha/2$. Otherwise, a black color is stored. $\Delta\alpha = (\varepsilon_\alpha - \alpha_{min})/n$, where ε_α and n are specified by the user. α_{min} is the minimum value of $\alpha(p)$ amongst all the pixels. Next, a distance transform [Jai88] is applied to each of $\mathbf{I}_{k,B}$ and a distance image, $\mathbf{I}_{k,D}$, is created (Fig. 5(c)). Each pixel in $\mathbf{I}_{k,D}$ stores the distance to the nearest black pixel. Metaballs are then generated at the locations of all white pixels of $\mathbf{I}_{k,B}$ (Fig. 5(d)). The metaballs are placed on the image plane at this stage. The radii of the metaballs are set to the distance stored in the distance image $\mathbf{I}_{k,D}$. By using the above method, the radii of the metaballs correspond to the thicknesses of clouds and are smaller than the size of the cloud cells. In addition, metaballs with similar radii are generated at the neighboring pixels with similar values of α.

Next, in order to make the synthetic clouds look similar to those in the input photograph, the center densities of the metaballs are determined so that the cumulative density at each pixel becomes the same as the cloud intensity $\beta(p)$. That is, the densities are determined by solving the following minimization problem:

$$Q = \sum_{p=1}^{N} \left(\beta(p) - \sum_{l=1}^{M} q_l f(r_{lp}) \right)^2 \rightarrow \min, \qquad (9)$$

where N is the number of pixels of the cloud image, M the number of metaballs, q_l the center density of metaball l, f is



(a) cloud image (b) binary images $\mathbf{I}_{k,B}$
(c) distance images $\mathbf{I}_{k,D}$ (d) generating metaballs
(e) optimizing center density (f) density distribution

Figure 5: *Modeling of altocumulus.*

the field function of the metaball, and r_{lp} the distance from the center of metaball l to pixel p.

For solving Eq. (9), we use the following simple method. The center density of each metaball is iteratively updated by the following equation.

$$q_l^{(i+1)} = \max\{0, q_l^{(i)} + \kappa(\beta(p) - \sum_{l=1}^{M} q_l^{(i)} f(r_{lp})\}, \qquad (10)$$

where $q_l^{(i)}$ is the center density of metaball l at ith iteration and κ is a user-specified constant. We use 0.01 as a value of κ in creating the examples shown in Section 7. The iteration process is terminated when $|Q^{(i+1)} - Q^{(i)}|$ is smaller than a specified threshold, where $Q^{(i)}$ is the value of Q using the metaball density at ith iteration. This approach does not always converge to an optimal solution but provides a good solution that is sufficient for our purpose. In order to speed up the computation, the above process is implemented by using CUDA. Fig. 5(e) shows an example of an approximated cloud image.

Finally, with a similar method to that used for cirrus clouds, the cloud plane is specified by the user, and the center positions of the metaballs are projected onto the plane. The radii of the metaballs are scaled in proportion to the distance from the viewpoint. After the projection process, a volume data is created by subdividing the bounding box of all the metaballs and by calculating a density at each grid point (Fig. 5(f)).

5.3. Cumulus

For modeling cumulus, the surface shape of the clouds is calculated first. Next, the density distribution inside the surface is generated.

To calculate the surface shape, we assume that the clouds are thinner at the boundary and thicker in the center. We also assume that the thickness of the clouds is the same if $\alpha(p)$ is the same. Based on these assumptions, the surface shape is calculated in the following way.

First, the cloud image is converted into a binary image with a user-specified threshold. That is, if $\beta(p)$ is greater than the threshold, a white color is stored in the binary image. Otherwise, a black color is stored. Next, the distance transform is applied to the binary image and medial axes are extracted [Jai88] (Fig. 6(a)). A medial axis is a pixel where the distance is the local maximum. In Fig. 6(a), the colors of the medial axes correspond to the distances: the blue color corresponds to zero and the red color to the longest distance. We use the distances at the medial axes as the thicknesses at the pixels. We also set the thicknesses to zero at the pixels where there are no clouds (i.e., $\beta(p) = 0$). The thicknesses at other pixels are determined by propagating the above thicknesses to other pixels. For this process, we use the method for colorization of a gray scale image by optimization [LLW04]. In our method, the thicknesses are determined by minimizing the following energy function.

$$J(T) = \sum_p \left(T(p) - \sum_{q \in A(p)} w_{pq} T(q) \right)^2, \qquad (11)$$

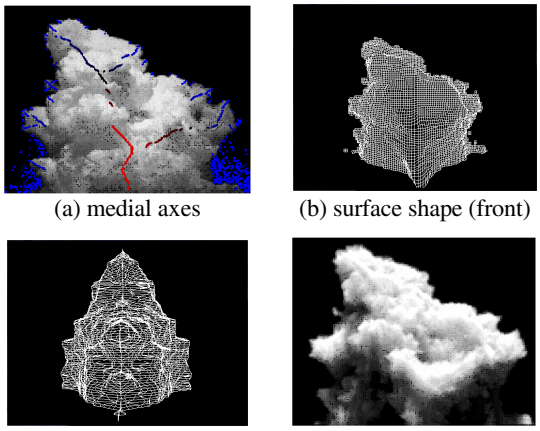
where T is thickness, p and q indicate pixel labels, and $A(p)$ represents a set of adjacent pixels of p. The weighting factor w_{pq} represents the similarity between p and q. The weighting factor is defined by:

$$w_{pq} \propto \exp(-(\alpha(p) - \alpha(q))^2 / (2\sigma_p^2)), \qquad (12)$$

where $\alpha(p)$ is the opacity of the clouds, represented by Eq. (6), and σ_p^2 is the variance of the opacities in $A(p)$. We use a GPU to solve the above minimization problem efficiently.

After the above process, the thickness of the clouds for each pixel is obtained and we can construct the surface shape of the clouds (Figs. 6(b) and (c)). As shown in Fig. 6(c), we assume the cloud shape is symmetric with respect to the image plane.

The density distribution inside the surface shape is generated by invoking the Perlin Noise [Per85]. However, the density of clouds typically becomes thin near the boundary. To take into account this fact, the density of clouds is generated in the following way. First, a binary volume is created by subdividing the bounding box of the surface shape into a regular grid. We assign a value of one to each grid point inside the surface shape and zero to an external grid point. Then, the three-dimensional version of the distance transform [BTG95] is applied to the binary volume. After the distance transform, each grid point stores the distance from the grid point to the nearest grid point outside the surface shape. Next, the distance is normalized so that the maximum distance becomes 1. We use the product of the Perlin noise and the normalized distance as the density at each grid point. Fig. 6(d) shows the density distribution obtained by the above process.



(a) medial axes (b) surface shape (front) (c) surface shape (side) (d) density distribution

Figure 6: *Modeling of cumulus.*

6. Parameter Tuning

This section describes the method for tuning the parameters involved in our system.

First, to create the cloud image, the threshold ε_c has to be specified for identifying cloud pixels (Section 4.1). This parameter is easily determined, since the identified cloud pixels are displayed in real-time.

Next, the user needs to specify the cloud plane. The camera parameters have also to be specified at this stage, since they affect the perspective distortion of the plane. If the actual camera parameters are available, we can use them. Otherwise, we determine them in the following way. We employ a pin-hole camera model and assume that the upward and the viewing directions of the camera coincide with the vertical (z axis) and the horizontal (x axis) directions, respectively. Then, the only unknown camera parameter is the viewing angle. Our system allows the user to interactively specify the viewing angle, together with the cloud plane. As for the cloud plane, the user can specify an arbitrary direction for the orientation of the plane. The user can also specify the size and the vertical/horizontal positions of the cloud plane.

For modeling altocumulus, the threshold ε_α and the number of binary images, n, (see Section 5.2) need to be specified. ε_α corresponds to the maximum opacity and this determines the pixels where metaballs are placed. While the user adjusts this parameter, pixels whose opacities are less than

Table 1: *Parameter settings. "fovy" is the viewing angle of the camera.*

Figure	image size	fovy	ε_c	ε_α	n
7	384×284	60°	0.4	N/A	N/A
8	452×300	60°	0.27	0.95	3
9	640×327	20°	0.27	N/A	N/A

ε_α are displayed in real time. Therefore, the user can easily determine this parameter. As for the number of the binary images, n, we determine this parameter experimentally and find that $n = 3 \sim 5$ works well for most cases.

7. Results

Figs. 7 through 9 shows three types of cloud generated by our method. In these figures, (a) shows the input photographs. In Figs. 8 and 9, images with different viewpoints are shown in (b) and (c). As shown in these examples, clouds that look similar to those in the photographs are generated. We use a desktop PC with Intel Corei7 (3.33 GHz) with NVIDIA GeForce GTX 295 to create these examples and the computation times were within 10 seconds.

The parameters used for the examples shown in this section are summarized in Table 1. These parameters are determined interactively, as described in Section 6. The time spent on determining these parameters ranged from one to three minutes. Most of the time was spent on tuning the cloud plane and the viewing angle; it ranged from 30 seconds to one minute. However, we verified that similar clouds were generated unless an extremely unnatural cloud plane was specified. Other parameters were determined within 30 seconds.

Fig. 10 shows two images of a synthetic scene including these clouds, rendered with different viewpoints and sunlight directions. Realistic images of clouds could be rendered.

8. Discussion

Our method can synthesize clouds that are similar to those in an input photograph. However, there is a case where the synthesized clouds are different from those that the user has in mind. Even in such a case, we believe that the user can create his/her desired clouds by editing the clouds generated by our method.

An obvious limitation is that our method does not work well in the case where multiple clouds overlap in the input photograph. In this case, our method treats them as a single cloud. This problem could be resolved by using multiple photographs.

The method using a chroma for identifying cloud pixels (see Section 4.1) works best for the photograph taken in the daytime, since the chroma of a cloud pixel is apparently different from that of a sky pixel. However, this method tends to fail when the sky becomes dark. To address this problem, a more sophisticated algorithm for the classification of the cloud pixels is required. Techniques for image segmentation could be applied [WC05] [LLW08].

(a) input photograph. (b) our result.

Figure 7: *Cirrus example.*

Another difficult situation for our method is the case where the intensities of clouds in the photograph are uniform. In this case, the opacities of the clouds calculated by our method become uniform. Since the shape of clouds is calculated by referring to the opacities, the thickness of the cloud shape also tends to become uniform, resulting in an unnatural shape of the clouds. This kind of situation occurs when an input photograph is taken at the sunset time, where clouds are uniformly dark.

9. Conclusion

We have proposed a method for generating clouds from a real photograph of clouds. Our method can synthesize three types of cloud: cirrus, altocumulus, and cumulus. Our method firstly computes the cloud image where intensities and opacities of the clouds are stored. For cirrus, the cloud image is used as a 2D texture. For altocumulus, metaballs are generated to define the 3D density distribution. For cumulus, the surface shape of clouds is calculated and the density distributions within the shapes are generated using Perlin noise. Our method can generate realistic clouds that are similar to the input photograph.

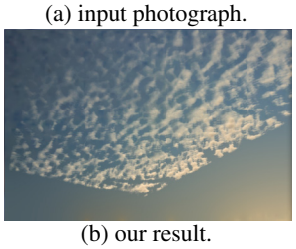
One important issue that should be addressed in the near future is that the shape of the cumulus clouds generated by our method is symmetric with respect to the image plane. Therefore, the clouds look unrealistic when viewed from the side. This problem could be addressed by adding random perturbations to the surface shapes calculated.

References

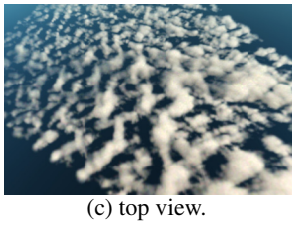
[BN02] BOUTHORS A., NEYRET F.: Modeling clouds shape. In *Proceedings of Eurographics 2004 (short papers)* (Aug. 2002).

[BTG95] BITTAR E., TSINGOS N., GASCUEL M.-P.: Automatic reconstruction of unstructured 3d data: Combining a medial axis and implicit surfaces. *Computer Graphics Forum 14*, 3 (1995), 457–468.

(a) input photograph.

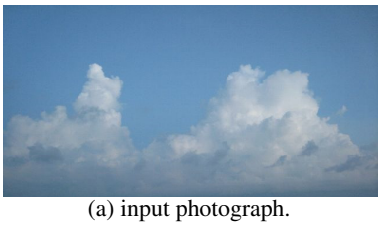


(b) our result.

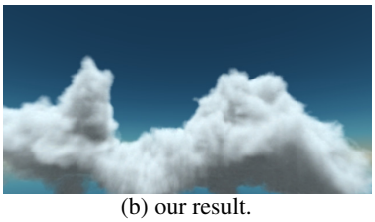


(c) top view.

Figure 8: *Altocumulus example.*



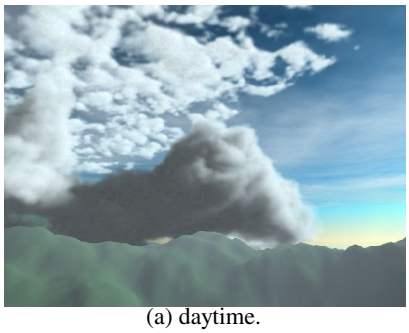
(a) input photograph.



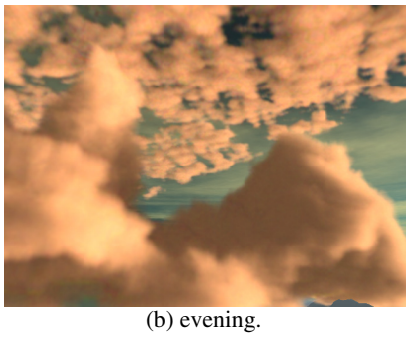
(b) our result.

(c) side view.

Figure 9: *Cumulus example.*



(a) daytime.



(b) evening.

Figure 10: *A cloud scene.*

[DKNY08] DOBASHI Y., KUSUMOTO K., NISHITA T., YAMAMOTO T.: Feedback control of cumuliform cloud formation based on computational fluid dynamics. *ACM Transactions on Graphics 27*, 3 (Aug. 2008), Article 94.

[DNYO98] DOBASHI Y., NISHITA T., YAMASHITA H., OKITA T.: Using metaballs to modeling and animate clouds from satellite images. *The Visual Computer 15*, 9 (1998), 471–482.

[Ebe97] EBERT D. S.: Volumetric modeling with implicit functions: A cloud is born. In *Visual Proceedings of SIGGRAPH 1997* (1997), p. 147.

[EMP*02] EBERT D. S., MUSGRAVE F. K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing & modeling: a procedural approach*. Morgan Kaufman, 2002.

[Gar85] GARDNER G. Y.: Visual simulation of clouds. *Computer Graphics (Proceedings of SIGGRAPH 1985) 19*, 3 (July 1985), 297–304.

[Har03] HARRIS M. J.: *Real-time Cloud Simulation and Rendering*. University of North Carolina Technical Report TR03-040, 2003.

[Jai88] JAIN A. K.: *Fundamentals of Digital Image Processing*. Prentice Hall, 1988.

[KH84] KAJIYA J. T., HERZEN B. P. V.: Ray tracing volume densities. *Computer Graphics (Proceedings of SIGGRAPH 1984) 18*, 3 (Aug. 1984), 165–174.

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 689–694.

[LLW08] LEVIN A., LISCHINSKI D., WEISS Y.: A closed form solution to natural image matting. *IEEE Transaction on Pattern Analysis and Machine Intelligence (TPAMI) 30*, 2 (2008), 228–242.

[MYND01] MIYAZAKI R., YOSHIDA S., NISHITA T., DOBASHI Y.: A method for modeling clouds based on atmospheric fluid dynamics. In *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications* (Aug. 2001), pp. 363–372.

[NND96] NISHITA T., NAKAMAE E., DOBASHI Y.: Display of clouds taking into account multiple anisotropic scattering and sky light. In *Proceedings of ACM SIGGRAPH 1996, Annual Conference Series* (1996), pp. 379–386.

[Per85] PERLIN K.: An image synthesizer. *ACM SIGGRAPH Computer Graphics 19*, 3 (1985), 287–296.

[PGB03] PEREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM Transactions on Graphics 22*, 3 (July 2003), 313–318.

[Sak93] SAKAS G.: Modeling and animating turbulent gaseous phenomena using spectral synthesis. *The Visual Computer 9*, 4 (1993), 200–212.

[SSEH03] SCHPOK J., SIMONS J., EBERT D. S., HANSEN C.: A real-time cloud modeling, rendering, and animation system. In *Proceedings of Symposium on Computer Animation 2005* (2003), pp. 160–166.

[Vos83] VOSS R.: Fourier synthesis of gaussian fractals: 1/f noises, landscapes, and flakes. In *SIGGRAPH'83: Tutorial on State of the Art Image Synthesis* (1983), vol. 10.

[WC05] WANG J., COHEN M. F.: An iterative optimization approach for unified image segmentation and matting. In *Proceedings of International Conference on Computer Vision (ICCV)* (2005), vol. 2, pp. 936–943.

[WT90] WYVILL G., TROTMAN A.: Ray-tracing soft objects. In *Proceedings of CG International'90* (1990), pp. 469–476.