JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 27, 891-913 (2011)

An Effective Framework for Cloud Modeling, Rendering, and Morphing*

CHUNG-MIN YU AND CHUNG-MING WANG+
Department of Computer Science and Engineering
National Chung Hsing University
Taichung, 402 Taiwan

In many multimedia applications such as computer games, flight simulation, and special effects for movies, clouds play an important role. In this paper, we present a variety of techniques for cloud modeling, cloud rendering, and cloud morphing in order to make the visual appearance of clouds in these applications more effective and interesting. For cloud modeling, we propose a polygon sampling technique that enables users to accurately model various types of cloud shapes from a 2D image. Moreover, if a 3D model is available, we provide a tetrahedron sampling method to model the cloud. In cloud rendering, we present two line integral convolution (LIC) techniques. The first technique eliminates the Gaussian blobs caused by the splatting algorithm that is commonly used in conventional cloud rendering while the second one produces a sky plane which increases the visual appearance of the cloud rendering. Finally, we present a "greedy" morphing algorithm for the purpose of cloud morphing. This algorithm allows a gradual transition of smooth cloud shapes during the animation process with only a limited number of frames. Our techniques increase the visual realism of the appearance of clouds.

Keywords: cloud modeling, cloud rendering, cloud morphing, line integral convolution, Gaussian blobs

1. INTRODUCTION

It is visually pleasing to have clouds in a virtual scene with a wide sky. As a result, many techniques have been devised for cloud rendering [1, 2]. Still, some problems remain unsolved. In this paper, we present an effective framework in order to accomplish our goal of producing clouds in a virtual scene. In particular, we provide three techniques for cloud modeling, cloud rendering, and cloud morphing.

In cloud modeling, we adopt the particle approach, and present a polygon sampling technique. By using this technique we obtain various 3D shapes of a cloud by first sampling the target 2D shape and then stretching it into 3D space. The 2D target shape can be a cloud image provided by a user. In addition, we can take into consideration a 3D polygon model where our framework can segment this polygon model into a number of tetrahedrons in which the position of the particles in three-dimensional space can be determined by a proposed tetrahedron sampling method.

In cloud rendering, we adopt the splatting technique, [2, 3], which normally maps Gaussian texture onto billboards. This technique has been widely adopted for most particle-based cloud systems in the computer graphics community. However, Gaussian textures

Received October 19, 2009; revised March 13 & May 11, 2010; accepted June 10, 2010.
Communicated by Tong-Yee Lee.
* This paper was partially supported by the National Science Council of Taiwan, R.O.C. under Grant No. NSC 98-2221-E-005-073-MY3.
+ Corresponding author.

have symmetric features in all directions causing an artificial Gaussian blob effect when using the traditional splatting algorithm for cloud rendering [3, 4]. In a recent article published by Bouthors *et al*. [4] this visually implausible effect is evident where some artificial non-cottony sharp effects are visible at the edge of the cloud shape, Fig. 1 (a). In this paper, we propose a line integral convolution (LIC) technique to solve the drawback of Gaussian textures. The LIC technique considers various gradients to produce LIC textures from an image given by users. Dissimilar to the Gaussian texture, the LIC texture is asymmetric and due to this significant feature more cottony effects are produced along the edge of the cloud, as shown in Figs. 1 (c) and (d). Besides, we also present a LIC texture database, which provides various LIC textures produced by controlling a variety of length and orientation when operating the LIC convolution process. This database enables our framework to allow different appearances in the cloud during the rendering process. In addition, the LIC technique can be used to extract the characteristics of a cloud photograph to form a LIC texture. This allows us to produce a similar cloud layer by mapping it to the sky plane, as shown in Fig. 14 (d), producing visually plausible results.

(a) Bouthors *et al*.'s method [4]. (b) Real cloud photograph. (c) Harris *et al*.'s method [3]. (d) Our technique.

Fig. 1. (a) The artificial non-cottony sharp effects are located at the edge of the shape (red circles); (b) A realistic cloud photography contains cottony edges and convoluted visual effects (red circles); (c) Artificial Gaussian blobs (red circle) along the edge of the cloud are noticeable using symmetric Gaussian texture in Harris *et al*.'s work; (d) The clouds become cottony in our system using an asymmetric LIC texture database.

The morphing effect is an interesting topic that has been studied in the computer graphics community, and is an effective animation method used to create special effects in the entertainment industry. For example, morphing techniques can make monotonous scenes in movies and games much more vivid and attractive. In user-friendly animation applications, morphing is useful as an important technique for animating key frames. All frames in an animation clip can be produced by the morphing technique as long as we provide key models. In this paper, we extend our investigation to cloud morphing. In particular, we propose a cloud morphing algorithm. To the best of our knowledge, we are the first to present an approach for cloud morphing by using the particle system. Our algorithm extends the modeling technique to determine the corresponding particle attributes during cloud morphing with the use of a "greedy" algorithm. Thus, our system can re-form the clouds that are rendered and transform the appearance of the clouds gradually from the source shape to the target shape. This provides very plausible visual results.

This paper is organized as follows. In section 2, we present an overview of cloud modeling and rendering. In section 3, we explain the proposed techniques, including cloud

modeling, cloud rendering, and cloud morphing. In section 4, we describe some experimental results and present some rendered images. Finally, in section 5, we present our conclusion and suggest future work.

2. RELATED WORKS

This section presents an overview of traditional cloud modeling, rendering, and morphing. Cloud modeling is designed as an effort to simulate cloud shapes for a 2D image or a 3D model. Cloud rendering, however, deals with the method of rendering a cloud image, given a cloud model, by using an appropriate illumination model. We are unaware of any professional literature for cloud morphing which uses the particle system. Instead, we will briefly present works about morphing from the image, polygon, and fluid communities.

2.1 Cloud Modeling

The reference literature indicates that the particle system [5] was first used to simulate many natural phenomena, such as fire and gas. Later, because of its simplicity and effectiveness, this system was adopted for cloud modeling [3, 6-9]. Dobashi *et al*. provided a method of modeling thin cloud layers with metaballs from satellite images by pixel calculation. However, it took much time to calculate the pixel relations. In this paper, a faster sampling method is applied to model clouds, and the thickness of the clouds is assigned by stretching them with a statistical method. As an alternative, Harris *et al*. [2] used the particle system to simulate clouds with real-time performance. Unfortunately, they focused on the rendering method, thus little information was given to detail their approach to modeling the cloud in their cloud simulation system. Instead, cloud modeling was considered as a "black box," where an irregular random distribution of particles was used in the system. Overby *et al*. [7] extended Harris *et al*.'s work and provided an approach to modeling cumulus clouds and stratus clouds. Wang [9] described a cloud system using particles to interactively model a dozen different cloud types. This system improves upon an earlier effort which could model only one type of cumulus cloud. Bouthors and Neyret [6] provided a method which used a hierarchy of quasi-spherical particles to model cumulus clouds.

The voxel method is an alternative approach for cloud modeling which was proposed to simulate stable fluids [10]. This approach models the clouds by using a number of grids where the status of a grid represents the attributes of a cloud. The voxel method provides the advantage of voxel smoothing and density calculation. It became a popular approach for producing the animation of clouds [2, 11, 12]. However, the method needs a great deal of computing time when the dimension of voxel increases. Dobashi *et al*. [11] provided a description for cloud modeling using the voxel method. Harris *et al*. [2] modeled clouds using the voxel method, and they used partial differential equations to describe the fluid motion. Liao *et al*. [12] modeled clouds by the voxel method where a look-up table is employed at run-time to increase the performance.

The procedural solid noise technique is another approach for cloud modeling [1, 13]. In this approach, shapes of the cloud are modeled by using several dedicated functions with appropriate parameters [14]. Unfortunately, it is difficult to find suitable functions

and proper parameters to produce a cloud with a desired shape and to control the change of the shape.

2.2 Cloud Rendering

The intention of cloud rendering is to render an image of a cloud using a particular algorithm and a specific illumination model. An early approach for cloud rendering was accomplished by a ray tracing algorithm [15]. However, the algorithm requires much computing time that is not suitable for a real-time system.

Cloud rendering using the splatting algorithm was first introduced by Nishita *et al*. [16]. This algorithm was popular due to its simplicity and performance [6, 8, 9, 11, 12]. Dobashi *et al*. [10] used this algorithm to render clouds in their voxel-based system. Liao *et al.* [12] provided a hierarchical texture caching technique to speed up the rendering performance. Harris *et al*. [2, 3] employed the splatting algorithm and achieved a real-time performance in their particle system.

A recent publication by Bouthors *et al*. [4] presents an approach to produce the effect of interactive multiple anisotropic scattering in clouds which simulates light transport inside a cloud. They use a complex structure to represent a cloud: a mesh is used to describe the outer cloud boundaries at a low resolution, and a procedural volumetric hypertexture adds details under the boundary up to a certain depth, until finally the core of the cloud is considered homogeneous. Their algorithm accounts for all kinds of light paths through the medium, and preserves the anisotropic behavior caused by various types of light paths. Rendering is done efficiently in a shader on the GPU. Unfortunately, an artificial Gaussian blobs effect is still generated in the rendering result. Although this side effect can be improved by using more meshes, this increases the workload for smoothing densities and calculating attributes. The workloads required then hamper the system in its effort to accomplish the performance of real-time rendering. Angelidis and Neyret [17] presented a simulation of smoke based on vortex filament primitives where ellipsoid blobs are adapted to simulate the smoke. However, the shapes of the ellipsoid blobs are close to a symmetric feature which still produces an artificial Gaussian blobs effect.

In this paper, we employ the particle system because it offers the advantage to achieve real-time performance. We propose a LIC technique which provides a great deal of flexibility in order to produce non-symmetric LIC textures by controlling the length and orientation in the line integral convolution process. The LIC texture allows us to eliminate the Gaussian blobs effect when using the splatting algorithm for rendering. Consequently, our rendering result produces cottony effects around the border edge of a cloud, as shown in Figs. 1 (b) and 12 (b).

2.3 Cloud Morphing

As we have mentioned before, the morphing effect is an important topic in the field of computer graphics for a variety of applications in movies, games, and animation applications. So far, there are many morphing techniques for images [18] and polygon models [19, 20]; however, this is not true for clouds. In a fluid system, some morphing techniques were provided for the physically-based voxel method, such as [21-23], but there are few articles in the literature about morphing in particle systems. Wu *et al.* did present a paper entitled

"particle morphing" [20]. To the best of our knowledge, however, their approach belongs to the polygon morphing. A basic assumption of their approach requires that the source and the target polygon contain exactly the same number of triangles, which can be achieved by the mesh subdivision process. Once this assumption is held, each triangle of the source object is mapped to that of the target, and each triangle of the source mesh is morphed to its corresponding one in the target mesh along a modulated user-defined path. Since the particle system has lower computation complexity than the voxel system, we believe it is worth investigating techniques to create cloud morphing.

In this paper, we introduce a technique which extends our modeling and rendering approaches for cloud morphing. The technique we present employs a greedy algorithm to determine the correspondence for a particle between the source and the target shape. In addition, using the approach of B-spline interpolation allows our algorithm to change shapes from the source shape to the target shape smoothly. Our technique considers morphing between particles in the cloud particle system. The proposed technique allows that the target shape has more particles than those of the source shape in order to provide the morphing flexibility. We will detail our proposed three techniques in the next section.

3. PROPOSED TECHNIQUES

This section presents our three effective techniques. First, we describe a new technique for cloud modeling, followed by techniques for cloud rendering and morphing. The flow chart of the proposed algorithms is shown in Fig. 2. The framework consists of three techniques, cloud modeling, rendering and morphing. First, given a 3D polygon model or a target 2D Shape as an input, our framework produces a cloud model using the proposed

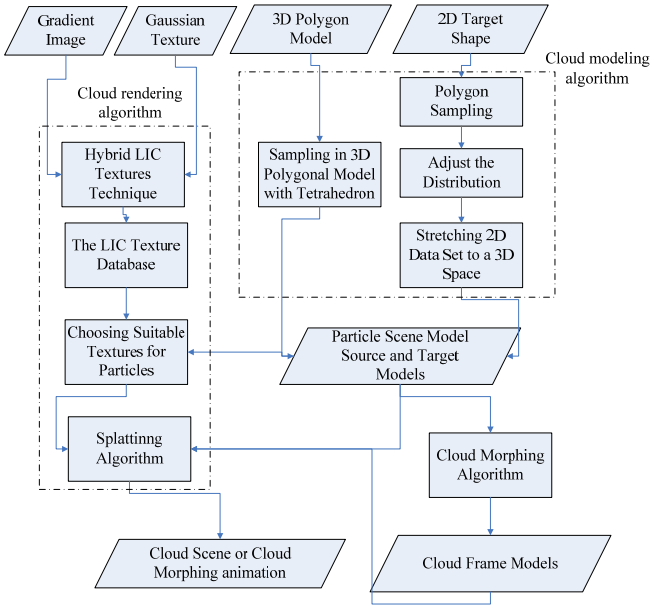


Fig. 2. The flowchart of the framework.

cloud modeling algorithms. In cloud rendering, an LIC texture database will be produced first using a hybrid LIC technique together with a user-drawn gradient image and Gaussian texture. Then, the cloud model will be rendered by referring to the constructed LIC texture database using the splatting rendering algorithm. Finally, in cloud morphing, two cloud models are considered as the source and target models. Then, the in-between frame models can be generated using the proposed cloud morphing algorithm. We detail each process in the following sections.

3.1 New Cloud Modeling Techniques

In this paper, we use the particle system for cloud modeling. We present the polygon sampling and tetrahedron sampling for cloud modeling. The first technique aims at effectively sampling the inputs of 2D shapes, and the second one aims at an effective sampling of the input of 3D models. Both sampling techniques determine the positions of particles, from which we can produce smooth densities of the particle. Once the attributes of particles are decided, the cloud model is rendered by the splatting algorithm. The following three subsections explain our approach of forming the cloud model and effectively making a smooth density of the particle for cloud rendering.

3.1.1 Polygon sampling

Given an input image containing a target shape, such as the bunny in black shown in Fig. 9 (a), we need to generate particles to represent this object in order to model a cloud. We achieve the goal of particle generation by the sampling technique. In the beginning, we transform the target shape to a polygon structure containing points and indexes. As shown in Fig. 9 (b), the polygon structure can be constructed by tracing out the contour of an image, and recording its points and indexes as the green line. We then begin sampling in this close region. Our method selects a point from the bounding rectangle of the polygon and uses the grid method [24] to decide whether or not the selected point is in the polygon. By the use of stratified sampling and repeating the above steps, we can sample a polygon with the target shape. Since we perform with stratified sampling, uniform sample points are distributed in the polygon. However, uniform distribution makes the edge of the cloud non-cottony. For this reason, we adjust the particle distribution by a heuristic method making the distribution tighter in the center than around the edge. Finally, we stretch these points produced in a 2D plane into a 3D space.

In the following sections we will explain the adjustment of the distribution, and the selection of the third coordinate for 3D. We also present a method for calculating smooth densities.

3.1.1.1 Adjusting the distribution

The point is uniformly distributed after the polygon sampling, as shown in Fig. 9 (a). Unfortunately, the uniform distribution leads to a drawback where the details of the target shape disappear even when we have generated large amount of particles; for example, the ears of the Bunny rabbit are difficult to visualize. To maintain the details we adjust the distribution in such a way that the distribution is tighter in the center but sparser around

the boundaries. Our distribution adjustment method is a heuristic technique using a random parameter (ε) in the range of [0, 1], which controls the shifting distance of a particle. First, we produce the medial axis from a 2D skeletonization algorithm [25], as shown in yellow lines inside the Fig. 9 (b). Then, each particle (p) finds the nearest point on the medial axis (q). We can compute the normal vector N_{pq} and the distance d_{pq} between the particle p and the point q. Finally, we move the original particle to its new position p' using Eq. (1), where $d_{\max}$ represents the maximum distance within the whole particles between a particle p and its associated point q.

$$p' = p + (d_{pd}/d_{\max}) \times \varepsilon \times N_{pq} \tag{1}$$

3.1.1.2 Stretching 2D data set to a 3D space

We produce two-dimensional points (x, y) in a 2D plane after adjusting the particle positions. We need to stretch them from a 2D plane to a 3D space in order to create three-dimensional points (x, y, z). A naïve approach is to assign the z-axis coordinates randomly within the maximum z-axis coordinates, $Z_{\max}$. However, this approach may result in two defects. The first defect is that a blob effect may appear around the edge of the cloud because there are as many particles around the edge as the center. The second defect is that this may construct a cloud with an unnatural shape. The nature of clouds reflects that a more reasonable distribution of a cloud model is that it is thicker in the center and thinner at the edge. Thus, we derive the z-axis coordinates by a non-uniform sampling technique. Let $R(x, y)$ be the z-range of a two-dimensional points (x, y) in the XY-plane. $R(x, y)$ can be represented using Eq. (2.1),

$$R(x, y) = D_{\max}(x, y) - D(x, y), \tag{2.1}$$

where $D(x, y)$ and $D_{\max}(x, y)$ are the distance and the maximum distance between (x, y) and the center (x_e, y_e). We then employ a probability density function (pdf), shown in Eq. (2.2) to obtain a cumulative distribution function (CDF), shown in Eq. (2.3). By giving a random number ε that is in the range of [0, 1], we can sample the z values as shown in Eq. (2.4).

$$f(z) = \frac{2}{R(x, y)^2}(R(x, y) - z) \tag{2.2}$$

$$F(z') = \int_0^{z'} \frac{2}{R(x, y)^2}(R(x, y) - z)dz \tag{2.3}$$

$$z' = R(x, y) - R(x, y) \times \sqrt{1 - \varepsilon} \tag{2.4}$$

In this way, a virtual 3D cloud model can be formed from points set in a 2D plane, and the positions of the particles in the cloud are all decided.

3.1.2 Sampling in 3D polygonal model with tetrahedron

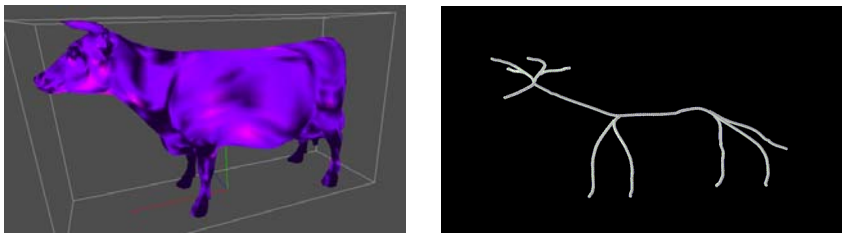
In addition to using a 2D shape to model clouds, it would be more convenient to have had a 3D model. When modeling clouds from 3D models, sampling in a polygonal model

is also a problem. As mentioned before, the main sampling distribution is tight at the center, and loose at the edge. For this purpose, we use a non-uniform sampling method to solve this problem. The point about this solution is that it becomes clear that each model can be divided into tetrahedrons from the points on the medial axis. It implies that as long as we can sample all of the tetrahedrons, the model is sampled too. The main advantage of the tetrahedron is that its sampling can be easily controlled. For example, our demand here is tighter at the apex and looser at the bottom, since all tetrahedrons use the points on the medial axis which is undoubtedly within the model as the apex. Thus, all we have to do is to find the medial axis of the model and sample them by the tetrahedron method. These detail issues are taken up in the next sections.

3.1.2.1 Extracting the medial Axis by skeletonization

Extracting the medial axis from a 3D model is a well-known problem of computer graphics. Thus, there are many existing solutions to this problem. The most useful and famous method is skeletonization. It is a technique that finds the diaphysis of a model with volume representation by thinning it. Recently, Cornea *et al.* [26] provided an algorithm to compute curve-skeleton, which is called the medial axis here, as Figs. 3 (a) and (b). Thus, in our method, we just only first voxelize [27] the model, and then extract the medial axis by computing the curve-skeleton. Next, the medial axis is transformed back to the coordinate axis of the model. There is no need to go into detail about voxelization and curve-skeleton here because these topics are discussed in previous literature [26, 27].

After extracting the medial axis, each triangle face takes the nearest point on it as the apex constructing a tetrahedron, and then, the tetrahedron sampling method is taken.



(a) The 3D cattle model. (b) The medial axis of the cattle model.

Fig. 3. Extracting the medial axis from a 3D model by the curve-skeleton.

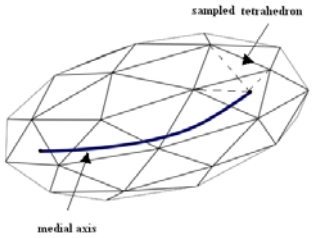


Fig. 4. Each triangle face finds the nearest point on the medial axis to establish a tetrahedron.

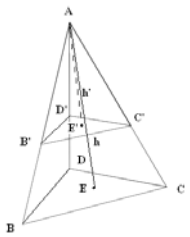


Fig. 5. The tetrahedron sampling chart. Samples are produced, $AE = h$.

3.1.2.2 Sampling in a tetrahedron

The next step reveals that for each tetrahedron in the model, the apex is on the medial axis. Since we extracted the medial axis in last section, each triangle face can establish a tetrahedron by connecting to the nearest point on the medial axis, see Fig. 4. It is now obvious that if all the tetrahedrons are sampled tighter at the apex and looser at the bottom, the model would be sampled tight at the center, and also loose at the edge.

For our purpose, we derive a CDF for sampling a tetrahedron by deciding the height h', as seen in Fig. 5. The h' can be formulized with respect to the height h of the tetrahedron, as shown in Eqs. (3.1) and (3.2).

$$CDF = \frac{\frac{h'^{3+n}}{h^2(3+n)}(\Delta BCD)}{\frac{h^{3+n}}{h^2(3+n)}(\Delta BCD)} = \frac{h'^{3+n}}{h^{3+n}} \tag{3.1}$$

$$h' = h^{n+3}\sqrt{CDF}, \begin{cases} n = 0, & \text{unform} \\ n = 1, & \text{linear increase} \\ n > 1, & \text{nonlinear increae} \\ n < 0, & \text{nonlinear decreae} \end{cases} \tag{3.2}$$

Eq. (3.2) shows when $n < 0$, the distribution of points decreases nonlinearly from the apex A to the bottom triangle BCD. When the height h' is derived, a unique triangle $\Delta B'C'D'$ is determined, allowing us to uniformly sample this triangle. Our demand for the distribution of samples is to produce more samples in apex A which is located at the inner part of a cloud, and to generate fewer numbers of samples at the bottom triangle ΔBCD.

One well-known approach is that of uniformly sampling a triangle as shown in Eq. (4),

$$P = B' + \varepsilon_2 \times \sqrt{\varepsilon_1} \times (C' - B') + (1 - \sqrt{\varepsilon_1}) \times (D' - B'), \tag{4}$$

where ε_1 and ε_2 are two random floating numbers between [0, 1].

As to the sampling number, a user-controlled density D is defined as the sampling number per voxel. Thus, in each tetrahedron, the sampling number SN is expressed by Eq. (5),

$$\mathrm{SN} = \frac{V_{pyramid}}{V_{vox}} \times D, \tag{5}$$

where $V_{pyramid}$ is the volume of the tetrahedron, and V_{vox} is the volume of each voxel. Note that although our system uses the particle system, we need to subdivide the 3D space of a given 3D polygon model into a number of desired voxels in order to produce the sampling numbers SN.

3.1.3 Smoothing the particle density

In addition to determining the positions of the particles, another attribute of particles needs to be computed in cloud modeling in order to render more realistic clouds. In previous papers about particle systems [1-3], densities are randomly assigned or procedurally controlled. The above methods calculate unexacting densities, but may result in an unnaturally granular appearance on the cloud surface.

For this reason, we present an approach to smooth particle densities. First, we construct a voxel structure by dividing the bounding box of the cloud model into $n \times n \times n/\sqrt{2}$ voxels. Second, using Eq. (6), we calculate the density of each voxel based on the number of particles it contains over the maximum particles in these $n \times n \times n/\sqrt{2}$ voxels.

$$V_d(i, j, k) = \sqrt{N(i, j, k)/N_{\max}} \tag{6}$$

In this equation, $V_d(i, j, k)$ represents the density of $V(i, j, k)$ we intend to derive, $N(i, j, k)$ indicates the number of particles inside this voxel, and $N_{\max}$ is the maximum particle numbers of all of the voxels.

Finally, we smooth the densities for each particle using a weighted function by considering the densities of the nearby voxels, as shown in Eq. (7):

$$p_{id} = \sum_{i, j, k \in \Omega(R)}^{N} V_d(i, j, k)\, f(|P_{ip} - V_p(i, j, k)|), \tag{7}$$

where P_{id} defines the density of the particle P_i, N is the number of voxels near P_i in a distance of R, P_{ip} and $V_p(i, j, k)$ defines the positions of P_i and $V(i, j, k)$, and f is a weight function dependent on the distance between P_i and $V(i, j, k)$, a simple case is the inverse proportion function. Using this smoothing method, the densities of particles may be closer to their neighbors.

3.2 New Cloud Rendering Techniques

Once cloud modeling is completed by the techniques stated in section 3.1, we are able to show the cloud on a screen by our cloud rendering technique, which we will explain in this section. Our method improves the splatting algorithm cited previously by the use of LIC (line integral convolution) [28]. This technique makes pixels of an image convoluted when following a given gradient image.

In the original splatting algorithm, cloudy Gaussian map textures are mapped on billboards as particles. However, if all the particles are mapped in that way, this will result in producing the side effects of the artificial Gaussian blobs shown in Fig. 1 (a). This is because the Gaussian textures have symmetric features in all directions.

For this reason, we use LIC to form various types of convoluted and asymmetric cloudy textures with different lengths and directions from the Gaussian texture, and then we map them on the billboards with a hybrid method. In this way, we can reduce the effects of the Gaussian blobs, and achieve more cottony and convoluted effects around the edge of the cloud, as shown in Fig. 1 (b).

Another advantage of LIC is that it can be used to produce a sky plane with a convoluted cloud layer from a realistic photograph of the cloud layer, as in Fig. 14 (c). The following section explains our new cloud rendering technique.

3.2.1 Cloud rendering

The splatting algorithm divides cloud rendering into two steps: the preprocess step and the process step. The former calculates the base colors of particles by a scattering theory; the latter maps textures with different densities and colors on the billboard.

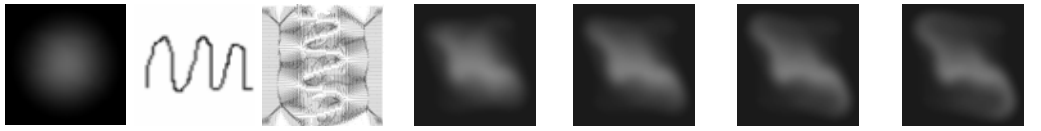
Our method of rendering a cloud with the splatting algorithm gives much better performance and offers better visual effects than other methods previously presented. With the preprocess step, shades under the simulated cloud layer absorb less sun light. This effect cannot be produced by the procedural texture method. The process step maps textures on the billboard by hardware performance. Given the abilities of the hardware, the performance rendering cloud can be real-time. However, such performance cannot be reached by the ray-tracing technique.

Since the splatting algorithm is better than the procedural texture in visual effect and the ray-tracing technique in its performance, we use the splatting algorithm. However, we improve its visual effect for cloud rendering.

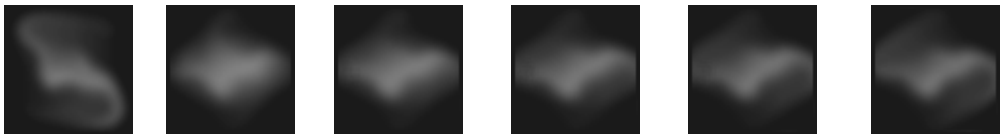
3.2.2 Hybrid LIC textures

As stated above, for reducing the Gaussian blobs and more cottony effects, we cannot use just a Gaussian texture for the splatting algorithm. Consequently, other kinds of textures should be used for mapping on billboards.

Our method produces textures by the LIC technique from the Gaussian texture. In this way, convoluted cloud textures are produced as indicated in Fig. 6. More obvious cotton effects are shown by mapping them on billboards; see Figs. 11 and 12.



(a) (b) (c) (d) Length = 20. (e) Length = 30. (f) Length = 40. (g) Length = 50.



(h) Length = 60. (i) Length = 20 with a rotation angle. (j) Length = 30 with a rotation angle. (k) Length = 40 with a rotation angle. (l) Length = 50 with a rotation angle. (m) Length = 60 with a rotation angle.

Fig. 6. (a) A symmetric Gaussian texture; (b) User-draw gradient images; (c) The visualized gradient image which demonstrates the feature of “normal to gradient”; (d)-(h) are produced LIC cloudy textures with different lengths; (i)-(m) are the textures adjusted by an image-moment based algorithm rotated by an upright angle.

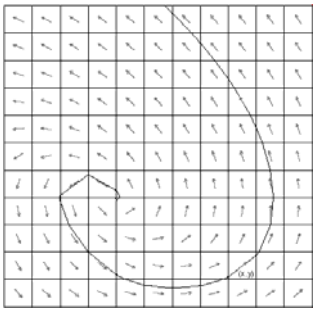


Fig. 7. A two-dimensional gradient image with different vector directions where the LIC is operated along the local stream line starting in cell (x, y) [28].

The original LIC technique takes three inputs, the gradient image, the source image, and the convoluted length. The gradient image dominates the gradient data of the output image; the source image dominates the pixel color of the output image; and the convoluted length allows users to control the degree of convolution. The LIC algorithm calculates an output image from the source image. Each output pixel is related to the corresponding vector of the gradient image and the vectors walking forward one grid and backward one grid, as in Fig. 7. This technique has already been introduced in the field of image processing [28]. Here, we use it to produce a convoluted cloud texture.

In our system, the gradient image is an input image drawn by users, and the source image is the Gaussian map texture. By controlling the lengths and the gradient image of the LIC technique, we can produce various user-controlled cloud textures, as shown in Fig. 6. The input Gaussian texture in Fig. 6 (a) acts as the source image, and Fig. 6 (b) is a user-drawn gradient image, while Fig. 6 (c) is the visualization of the vector distribution in Fig. 6 (b). Figs. 6 (d)-(h) indicates the result after the LIC operation with different lengths. In addition, for a more suitable cloudy texture, we smooth the LIC results with some iteration.

(1) The LIC Texture Database

In opposition to the Gaussian texture, the LIC texture is an asymmetric shape. Thus, it is necessary to induce some control on the texture's direction and LIC length, and we build the LIC texture database to do this. After producing each length of LIC texture, we rotate it to the upright angle as the 0° texture. The upright angle can then be calculated automatically by the image-moment based algorithm [29]. By this method, each LIC texture with different lengths can be calculated by an object orientation, θ, and then the upright angle here is $-\theta$. The 0° LIC texture can be derived by rotating the LIC texture, $-\theta$, shown as Figs. 6 (i)-(m). From the 0° LIC texture, 24 textures with different angles 0 – 360° can be made by rotating 15° for each iteration. In this way, the LIC texture database with about 24* length textures is built.

(2) Choosing a Suitable Texture for Particles

For each particle, we assign the LIC length and texture direction as indexes to choose a suitable texture.

When choosing the most suitable LIC texture for a particle, both the LIC length and

the LIC texture direction must be taken into account. First, in view of the length of the LIC, it becomes apparent that the further away from the center of the model that the particle is, the longer the LIC needs to be in order to produce cottony effects. In this way, the edge of the cloud would be seen as cottony and non-Gaussian blobs.

Thus, the length of each particle can be written as Eq. (8):

$$P_l = Min_l + (\frac{|P_{pos} - Cen|}{Maxdis})^2 (Max_l - Min_l), \tag{8}$$

where P_l is the particle's LIC length, Min_l is the user-defined minimum LIC length, P_{pos} is the position of the particle, *Cen* is the core of the cloud, *Maxdis* is the maximum distance between the core and the particles, and Max_l is the user-defined maximum LIC length. By using Eq. (8), the length of the particle will be longer at the edge and shorter at the center of the cloud.

Second, the texture direction of the particle can be decided by the relation between position of the particle and the center also. Thus, the texture direction of each particle θ can be written as Eq. (9):

$$\theta = \cos^{-1} \frac{Proj(P_{vecx})}{\sqrt{Proj(P_{vecx})^2 + Proj(P_{vecy})^2}}, \tag{9}$$

where P_{vecx} and P_{vecy} are the vector between the center and the particle. The *Proj*() function calculates the projecting coordinate of the vector to the view plane. This can be implemented by OPENGL function quickly. After deciding the length and texture direction of each particle, the LIC texture can be picked from the texture database quickly.

3.2.3 LIC cloudy sky plane

Another use of LIC in cloud rendering is to produce a cloudy sky texture which is mapped on a far sky plane. Previously, unnatural sky planes have been produced by procedural texture techniques because the functional control could not simulate a realistic sky well.

Now, we can select a method to produce a realistic sky plane from a picture of a cloud scene by using the LIC technique. First, we utilize a photograph of a real cloud scene as a source image. Secondly, we process the picture by using the image matting algorithm which is canvassed in [30]. This produces a texture with the cloud scene trait, see Fig. 14 (b). However, this texture still does not produce a good cloudy sky plane because of its sharp edge. Instead, we use the LIC technique which causes the texture to be more convoluted, thus generating a more naturally fluid effect. The non-convoluted sky plane is the input as a source image in the LIC technique, and the most suitable gradient image is the photograph itself. Since we have both the source image and the gradient image, the LIC technique can be adopted for them. Fig. 14 (c) is an example demonstrating an LIC cloudy sky plane, while Fig. 14 (d) shows the result of mapping the cloudy sky plane to a sky scene. By using the LIC sky plane, we reproduce the effects of far away cloud layers in a sky scene.

3.3 A New Cloud Morphing Algorithm

By using our modeling and rendering methods, it is easy to achieve cloud morphing effects since cloud morphing is a technique that re-shapes one cloud model to another. For this purpose, we need to have two cloud models, a source model and a target model with equal numbers of particles first. Both of them can be produced easily by our cloud modeling method, and the cloud morphing algorithm can be started. All we have to do is deform the source to the target model.

3.3.1 Correspondence of the source and target models

We need to find the correspondence between the particles in the source and target cloud models, because inappropriate correspondence may produce an unpleasant visual morphing effect. A good frame model ensures that in both the source and target models the traits are similar. In this paper, we provide a "greedy" algorithm to accomplish this goal. In our method we compare the source model with the target model in order to determine the pair of points which has the shortest path, as described below.

First, for each particle number i in the source model PS_i we find the particle number j in target model PT_j that has the minimum distance (PS_i, PT_j). Then, we exchange attribute PS_i and PT_j, where $\{PS_1, PS_2, \ldots, PS_n\} \in PS$, $\{PT_1, PT_2, \ldots, PT_n\} \in PT$. For example, if we assume $n = 5$, this means that the source group $\{PS_1, PS_2, PS_3, PS_4, PS_5\} \in PS$ and the target group $\{PT_1, PT_2, PT_3, PT_4, PT_5\} \in PT$. For PS_1, assume that (PS_1, PT_4) has the minimum distance. Then, the attribute of PT_1 and PT_4 will be exchanged in PT, and this will produce a new sequence $\{PT_4, PT_2, PT_3, PT_1, PT_5\}$ in the target group. Once all the particles in the source group are processed, we can produce a corresponding sequence between the source group and the target group.

After handling the particles as above, particles in the target model will automatically correspond to the source model with the shortest distance. This process is the greedy method of finding the correspondence with the shortest distance between the entire source model and the target model using a number of stages. The greedy method chooses a local optimum at each stage. By using this method, we can obtain a potential optimum mapping in the time complexity of only $O(n^2)$. Optimum mapping may exist, and we need to find it by a brute force method; however, the time complexity, which is $O(n \times n!)$, is much higher.

The method mentioned above is only suitable if the number of points of the source model is equal to that of the target model. In other words, it is a one-to-one correspondence between both of them. However, not every model has an equal number of points. The bigger the cloud is, the more points need to be sampled in order to describe it. To improve this greedy method, we propose a way of using this method in a reiterative manner in order to find the one-to-multiple or multiple-to-one correspondence.

Our method assumes both the particle set of the source model $\{PS_1, PS_2, \ldots, PS_m\} \in PS$, and the particle set of the target model $\{PT_1, PT_2, \ldots, PT_n\} \in PT$, where $m < n$, and $n = a \times m + k$, and where $k < m$. At first, $\{PS_1, PS_2, \ldots, PS_m\}$ finds a correspondence with $\{PT_1, PT_2, \ldots, PT_n\}$ by this algorithm, and the members of the set automatically fall into rank as $\{PT_1, PT_2, \ldots, PT_m\}$. After that, a reiterative action between the set $\{PS_1, PS_2, \ldots, PS_m\}$ and $\{PT_{m+1}, PT_{m+2}, \ldots, PT_n\}$ finds the final set $\{PT_{m+1}, PT_{m+2}, \ldots, PT_{2\times m}\}$. Here, an element in $\{PT_{m+1}, PT_{m+2}, \ldots, PT_{2\times m}\}$ finds its group, and maps to identical elements in $\{PS_1,$

PS_2, …, PS_m}. By repeating this kind of action a times, {$PT_{1\times m+1}$, $PT_{2\times m+1}$, …, $PT_{(a-1)\times m+1}$} automatically is a group mapping to the identical element PS_1. Similarly, elements in the set {$PT_{1\times m+z}$, $PT_{2\times m+z}$, …, $PT_{(a-1)\times m+z}$} map to the identical element PS_z, where $1 < z < a - 1$. Finally, the remaining elements do not know that their group in PT is {$PT_{a\times m+1}$, $PT_{a\times m+2}$, …, $PT_{a\times m+k}$}. Reverse mapping can be introduced here. Elements in the set {$PT_{a\times m+1}$, $PT_{a\times m+2}$, …, $PT_{a\times m+k}$} can find their group by using greedy algorithm mapping from the target set {$PT_{a\times m+1}$, $PT_{a\times m+2}$, …, $PT_{a\times m+k}$} to the source set {PS_1, PS_2, …, PS_m} and add themselves to their groups. In this way, a one-to-multiple correspondence can be found.

When, or if, $m > n$, there is a simple solution: the method above can be introduced by reversing itself. The mapping can go from PT to PS, instead of PS to PT. In this way, a multiple-to-one correspondence can also be found.

3.3.2 Cloud morphing technique

Since we have found a correspondence between the source and target model, a morphing technique should be proposed in order to move the source positions to the target positions. An easy morphing technique of linear interpolation can be applied to do this. However, if all the particles move by a linear route, uneven lengths of the paths of particles will happen. The result would be that some particles would be almost motionless, while some others would move far between neighboring frames. Because of this, it would waste more frames to make the morphing smooth. For this reason, a non-linear interpolation using B-Spline curve is provided here to make the lengths of the particles' paths stable. A B-Spline is a curve that fits generated data. Therefore, if the positions at the first frame (source positions), at the last frame (target positions), and at the middle frame (refining points) are known, all the other positions at other frames can be calculated by a B-Spline curve. The point about our method is that the shorter the path of the particle, the farther refining point we give it.

Thus, for each particle, we calculate a refining point based on its shortest path length. The refining point RF can be shown as Eqs. (10.1)-(10.3):

$$RF = \frac{S+T}{2} + \overrightarrow{ST}_{\perp} \times h, \tag{10.1}$$

$$\overrightarrow{ST}_{\perp} = Normalize(rndx, rndy, \overrightarrow{ST}(z) - (\overrightarrow{ST}(x) \times rndx + \overrightarrow{ST}(y) \times rndy)), \tag{10.2}$$

$$h = rnd(Maxdis - dis), \tag{10.3}$$

where S is the source position, T is the target position, $ST_{\perp}$ is a calculated vector perpendicular to the vector $\overrightarrow{ST}$, h is the calculated distance between RF and $\overline{ST}$, $rndx$ and $rndy$ are two random floating numbers in the range of [0, 1], and the function $rnd(i)$ means that an entry i of this function will produce the desired random values in the range of [0, i]. The term dis represents the distance between the source and the target positions. The term $Maxdis$ is the maximum distance of all particle pairs, and the sketch of our idea is shown in Fig. 8.

After getting the refining points, the B-Spline curve can be used to interpolate all positions at each frame. As to the calculation, we decide F by a statistical method for a smoother transformation, as shown in Eq. (11)

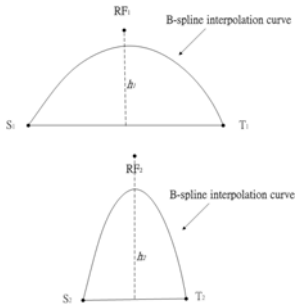


Fig. 8. B-spline interpolation diagram. For balance the distances, the nearest correspondence points have longer *h*.

$$F = \frac{(dis_{mean} + 2 \times dis_{var})}{dis_{smooth}}, \tag{11}$$

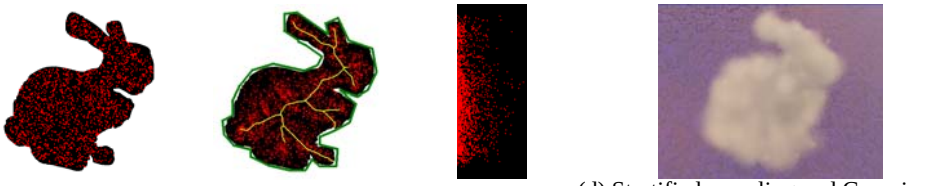
where dis_{mean} defines the mean of the distance between all pairs of corresponding particles, dis_{var} defines the variance of distance between all pair correspondence points, and dis_{smooth} is a user-controlled parameter meaning the longest distance the particles can move smoothly between neighboring frames.

In this way, frame models can be produced with their frame number. All we have to do now is to render them by our rendering method, and the cloud morphing effect is realized by our cloud morphing techniques.

4. RESULTS AND COMPARISON

In this section, we present results about cloud modeling, cloud rendering, and cloud morphing. Our experimental results were conducted on a PC with 2.8 GHz processor and 128 MB display card.

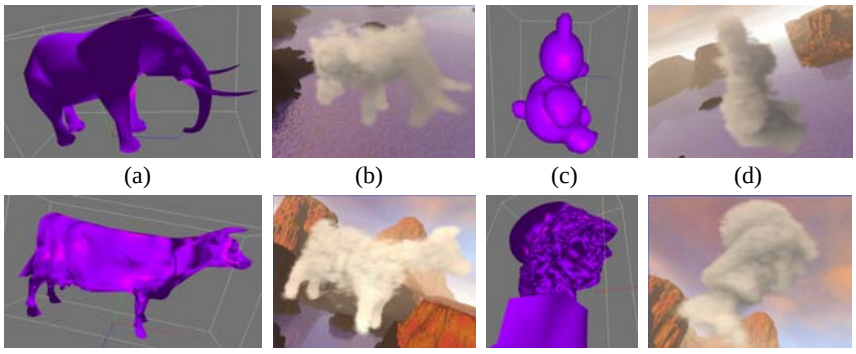
In cloud modeling, we tested cloud models using polygon sampling. Fig. 9 (a) shows the stratified sampling result. Fig. 9 (b) shows our polygon sampling result where the yellow line is the skeleton, and Fig. 9 (c) shows the YZ-plane sampling result that is produced for particles. There were 8000 particles in the model, and it took 0.36 seconds to sample the model. Fig. 9 (d) shows the rendered result before the particle adjustment where we employed the conventional Gaussian texture for rendering. As expected, the Gaussian blobs effect is visualized and the details of the ears are not preserved. Fig. 9 (e) shows the rendered result using stratified sampling for modeling, but employs the proposed LIC textures technique for rendering. We find the Gaussian blobs effect has disappeared. However, the details of the shape cannot still be preserved due to the stratified sampling approach. Fig. 9 (f) shows the rendering result using our sample adjustment for modeling but using conventional Gaussian textures for rendering. In contrast to Fig. 9 (e), Fig. 9 (f) preserves the details due to the contribution of our particle adjustment strategy. However, the Gaussian blob effect is still visible. Finally, Fig. 9 (g) shows the rendered result using the particle adjustment for modeling and the LIC texture for rendering. This is the final result produced in our system: it preserves the details by using the particle adjustment strategy and



(a) (b) (c) (d) Stratified sampling and Gaussian texture.

(e) Stratified sampling and LIC textures. (f) Adjusted sampling and Gaussian textures. (g) Adjusted sampling and LIC textures.

Fig. 9. Cloud modeling from a binary image (a) XY-axis polygon sampling result; (b) XY-axis adjusted sample result with skeleton; (c) YZ-stretched sampling result; (d)-(g) rendering results.



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 10. Model clouds from 3D models (up to bottom: Elephant, Teddy, Cow, Ateneal).

Table 1. Cost of our cloud modeling method.

Model name	Triangle faces	Sampling points	Sampling time (sec)	Performance (FPS)
Elephant	979	9,404	0.28	60-110
Teddy	1,595	13,492	0.33	60-110
Cow	2,920	15,909	0.56	60-110
Ateneal	9,394	34,832	1.32	30-80

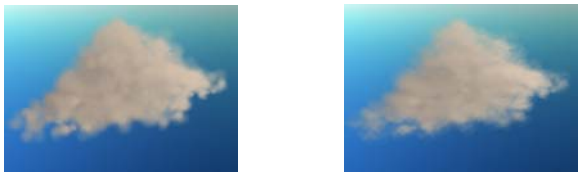
eliminates the Gaussian blobs effect by using the LIC texture. Our system demonstrates the real-time performance by achieving the least frame rate of 90fps.

Fig. 10 demonstrates that four cloud models can be produced in our system by using 3D polygonal models, Elephant, Teddy, Cow and Ateneal. The sampling costs are detailed

in Table 1. Similarly, with the aid of our sampling method and the LIC technique, the details are preserved and the Gaussian blobs are unseen.

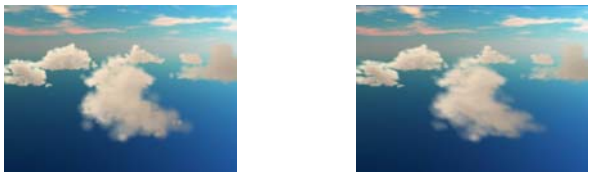
In cloud rendering, we first compare the rendering result of the cloud model "single-cloud," which is provided in Harris' skyworks [3]; see Fig. 11. Both the rendering performances reach real-time with a speed of 110fps and above. However, there are obvious artificial Gaussian blobs in Fig. 11 (a), while there are cottony effects in Fig. 11 (b). We also render a cloud scene call "small sky" in the Harris' skyworks with both the traditional splatting algorithm by Harris [3] and our hybrid LIC technique as shown in Fig. 12. There are some cloud details in Fig. 13. Both rendering performances reached real-time with a speed of 50fps and above. However, this makes it clear that our hybrid LIC technique has no artificial Gaussian blob side effects, which is obvious in the results of Harris.

Fig. 14 shows the results of the LIC cloudy sky plane. Fig. 14 (a) is a photograph of a cloud scene, and Fig. 14 (b) shows the results of extracting cloud traits by our method, while Fig. 14 (c) shows the cloudy sky plane after the LIC process. Finally, Fig. 14 (d) is the result of mapping the sky plane to form a scene.



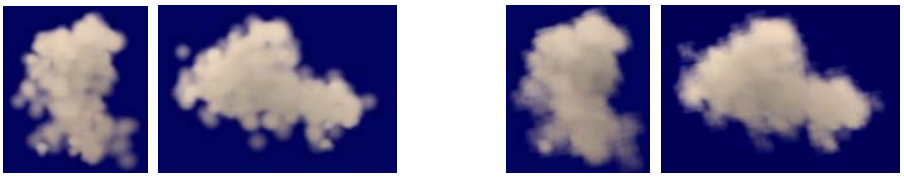
(a) (b)

Fig. 11. Single cloud model rendered with (a) Harris method using Gaussian textures produces Gaussian blobs and (b) our method using the hybrid LIC textures generates clouds with the cottony effect.



(a) (b)

Fig. 12. A comparison of the rendered scene using (a) Harris method and (b) our method. When the view is on top of the clouds, there are many artificial Gaussian blobs using Harris *et al.*'s method, but they become cottony using our hybrid LIC texture approach.

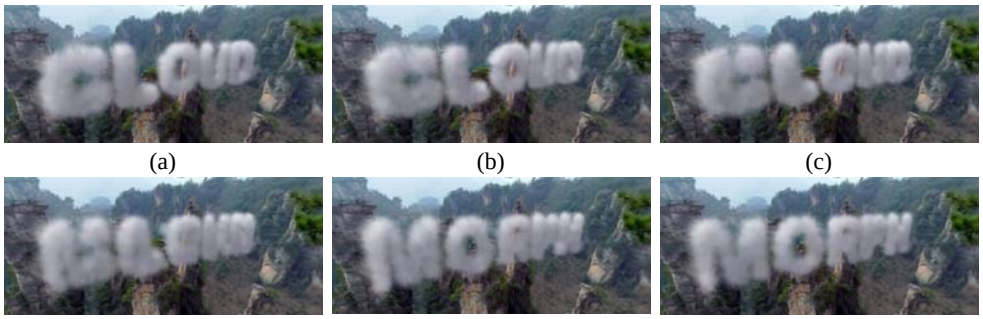


(a) Harris *et al.*'s method [3]. (b) Our hybrid LIC textures.

Fig. 13. Cloud details in the rendered scene with (a) Harris *et al.*'s method (Gaussian textures) and (b) our hybrid LIC textures. There are many artificial Gaussian blobs in (a), but in (b) the clouds are cottony.

(a) (b) (c) (d)

Fig. 14. (a) Real cloud photograph (b) extracted cloudy trait (c) LIC cloudy sky plane (d) LIC cloudy sky plane mapped to a sky scene.



(a) (b) (c)

(d) (e) (f)

Fig. 15. Cloud morphing from (a) the cloud font "CLOUD" to (f) the cloud font "MORPH."

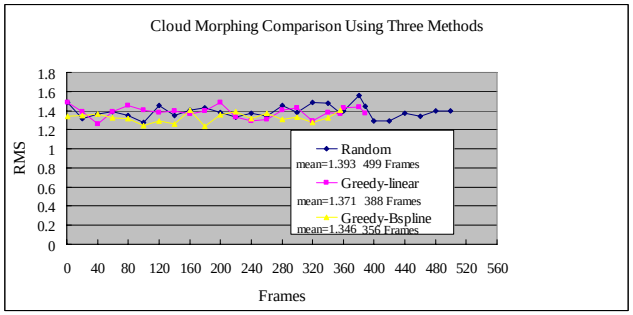


Fig. 16. The root mean square (RMS) error between two consecutive frames.

In the cloud morphing algorithm we implement an example morphing from the source model, appearing as the word "CLOUD," to the target model, appearing as the word "MORPH." There are 10,000 particles in the source and 15,000 particles in the target model, and 356 frames models are produced in 29.453 seconds by using the greedy algorithm and the B-Spline interpolation. Re-forming processes are shown in Fig. 15 and the attached file "01-Greedy Corresponding.avi." For comparison, a morphing result using the random particle correspondence is also provided as the attached file "02-Random Corresponding.avi." It is evident that the morphing behavior is made smoother by the greedy algorithm because of the shortest path correspondence between both models. Besides, the two morphing results using the Bspline Interpolation (03-Bspline Interpolation.avi) and the linear-interpolation (04-Linear Interpolation.avi), as shown as the attached files. We

find that the linear interpolation produces more superfluous frames than the B-Spline one. In Fig. 16 the root mean square (RMS) errors are indicated between two consecutive frames. It is clear that the approach of using the Greedy correspondence and the Bspline interpolation (yellow line) produces more stable variation than the other two approaches.

5. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented three techniques for cloud modeling, cloud rendering, and cloud morphing. Two sampling techniques are proposed for cloud modeling, the LIC technique is presented for cloud rendering, and the greedy morphing algorithm is developed for cloud morphing.

Given an image, a 2D polygon can be produced. By sampling this 2D polygon and stretching samples into 3D space, we produced a number of cloud particles with the desired shapes in a short time. In addition, our system can model a cloud using a 3D polygon model. By segmenting the 3D model into a number of tetrahedrons, we can operate the proposed tetrahedron sampling method in order to produce cloud particles that are tighter in the center but sparser around the edge.

We presented two approaches for cloud rendering. The first is the LIC texture database, and the second is the LIC cloudy sky plane. The LIC texture database produces LIC textures using a different length and orientation. When rendered by the splatting algorithm, the LIC texture database reduces the visually implausible Gaussian blob effect that is visualized when using the conventional Gaussian texture. In addition, the LIC texture database produces various kinds of cloud appearances by using different gradient images. The LIC cloudy sky plane produces the sky with texture from a real cloud photograph and this increases the visual realism of the cloud rendering.

Finally, we provided a cloud morphing algorithm which has not been reported in previous literature. Our algorithm determines particle correspondence in the shortest path between the source and the target models. We adopt the B-Spline interpolation rather than the linear interpolation to stabilize the distance between frames. This ensures a smooth transformation for cloud morphing, producing coherent morphing effects.

In conclusion, experimental results demonstrate that our proposed techniques increase the visual realism of the cloud appearance, making the results more effective and interesting.

In the future, we intend to develop an alternative approach for stretching particles in 2D to a 3D space. Another possible work is to find an even better correspondence technique for cloud morphing.

REFERENCES

1. G. Y. Gardner, "Visual simulation of clouds," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 19, 1985, pp. 297-304.
2. M. J. Harris, W. V. Baxter, T. Scheuerman, and A. Lastra, "Simulation of cloud dynamics on graphics hardware," in *Proceedings of Graphics Hardware*, 2003, pp. 92-101.

3. M. J. Harris and A. Lastra, "Real-time cloud rendering," *Computer Graphics Forum*, Vol. 20, 2001, pp. 76-84.
4. A. Bouthors, F. Neyret, N. Max, E. Brunet, and C. Crassin, "Interactive multiple anisotropic scattering in clouds," in *Proceedings of ACM Symposium on Interactive 3D Graphics and Games*, 2008, pp. 173-182.
5. W. T. Reeves, "Particle system – A technique for modeling a class of fuzzy objects," *ACM Transactions on Graphics*, Vol. 2, 1983, pp. 359-376.
6. A. Bouthors and A. Neyret, "Modeling clouds shape," *Computer Graphics Forum* 2004, pp. 1-4.
7. D. Overby, Z. Melek, and J. Keyser, "Interactive physically-based cloud simulation," in *Proceedings of the 10th Pacific Conference on Computer Graphics Applications*, 2002, pp. 469-470.
8. N. Wang, "Realistic and fast cloud rendering in computer games," in *Proceedings of ACM SIGGRAPH Conference on Sketch and Applications*, 2003, pp. 27-30.
9. N. Wang, "Realistic and fast cloud rendering," *Journal of Graphics Tools*, Vol. 9, 2004, pp. 21-40.
10. J. Stam, "Stable fluids," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 1999, pp. 121-128.
11. Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita, "A simple, efficient method for realistic animation of clouds," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, 2000, pp. 18-28.
12. H. S. Liao, T. C. Ho, J. H. Chung, and C. C. Lin, "Fast rendering of dynamic clouds," *Computer and Graphics*, Vol. 29, 2005, pp. 29-40.
13. K. Perlin, "An image synthesizer," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 19, 1985, pp. 287-296.
14. J. Schpok, J. Simons, D. S. Ebert, and C. Hansen, "A real-time cloud modeling, rendering, and animation system," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003, pp. 160-166.
15. J. Kajiya and B. V. Herzen, "Ray tracing volume densities," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques*, 1984, pp. 165-174.
16. T. Nishita, Y. Dobashi, and E. Nakamae, "Display of clouds taking into account multiple anisotropic scattering and sky light," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques*, 1996, pp. 379-386.
17. A. Angelidis and F. Neyret, "Simulation of smoke based on vortex filament primitives," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005, pp. 87-96.
18. D. Ruprecht and H. Muller, "Image warping with scattered data interpolation," *IEEE Computer Graphics and Applications*, Vol. 15, 1995, pp. 37-43.
19. T. Michikawa, T. Kanai, M. Fujita, and H. Chiyokura, "Multiresolution interpolation meshes," in *Proceedings of the Pacific Conference on Computer Graphics Applications*, 2001, pp. 60-69.
20. W. Wu, X. Jin, Y. Zhao, J. Feng, and Q. Peng, "Particle morphing," in *Proceedings of the 8th International Conference on CAD/Graphic*, 2003, pp. 1-6.
21. O. K. C. Au, C. L. Tai, H. K. Chu, D. Cohen-Or, and T. Y. Lee, "Skeleton extraction by mesh contraction," *ACM Transactions on Graphics*, Vol. 27, 2008, pp. 1-12.

22. A. Treuille, A. McNamara, Z. Popovic, and J. Stam, "Keyframe control of smoke simulations," *ACM Transactions on Graphics*, Vol. 22, 2003, pp. 716-723.
23. Y. S. Wang and T. Y. Lee, "Curve skeleton extraction using iterative least squares optimization," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 14, 2008, pp. 926-936.
24. E. Haines, "Point in polygon strategies," *Graphics Gems IV*, P. Heckbert, ed., Academic Press, Boston, 1994, pp. 24-46.
25. M. Foskey, M. Lin, and D. Manocha, "Efficient computation of a simplified medial axis," in *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2003, pp. 96-107.
26. N. D. Cornea, D. Silver, X. Yuan, and R. Balasubramanian, "Computing hierarchical curve-skeletons of 3D objects," *The Visual Computer*, Vol. 21, 2005, pp. 945-955.
27. M. Sramek and A. Kaufman, "Alias-free voxelization of geometric objects," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, 1999, pp. 251-266.
28. B. Cabral and L. Leedon, "Imaging vector fields using line integral convolution," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, 1993, pp. 263-272.
29. M. Shiraishi and Y. Yamaguchi, "An algorithm for automatic painterly rendering based on local source image approximation," in *Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*, 2000, pp. 53-58.
30. A. Bouthors, F. Neyret, and S. Lefebvre, "Real-time realistic illumination and shading of stratiform clouds," in *Proceedings of Eurographics Workshop on Natural Phenomena*, 2006, pp. 41-50.

Chung-Min Yu (游宗旻) received the B.S. degree in Computer Science and Information Management, Soochow University, Taiwan, in 2001, and the M.S. degree in Computer Science and Engineering, National Chung Hsing University, Taiwan, in 2003, respectively. He is currently a Ph.D. candidate in Computer Science and Engineering, National Chung Hsing University, Taichung, Taiwan. His research interests include computer graphics, multimedia, high dynamic range imaging, and information hiding algorithms.

Chung-Ming Wang (王宗銘) received the B.S. degree in Applied Mathematics from the National Chung Hsing University, Taiwan, in 1984, and the Ph.D. degree in Computer Science from the University of Leeds, Leeds, U.K., in 1992. From August 1986 to August 1988, he was a systems analyst in the President Enterprise Cooperation and the PAL Company. He is currently a Professor at the Department of Computer Science and Engineering, Institute of Networks and Multimedia, National Chung Hsing University, Taiwan. His research interests include computer graph-

ics, color science, virtual reality, multimedia systems, and three dimensional watermarking and steganography. Dr. Wang has won three Dragon Thesis Awards, funded by the Acer Computers, and Outstanding Paper Awards from the Computer Society of the Republic of China. He is a member of ACM, IEEE Computer Society and Eurographics.