An Implementation of a Simple, Efficient Method for Realistic Animation of Clouds

Cansın Yıldız

Dept. of Computer Engineering
Bilkent University
Ankara,Turkey
cansin@cs.bilkent.edu.tr

***Abstract*—I implement a Cloud Simulation suggested by *Dobashi et al.* at [1]. In this work, they propose a simulation process that results binary output of volumetric data, which is then smoothed using gaussian filter and rendered.**

I. Introduction

THIS report is a detailed presentation of achieved research on *a Simple, Efficient Method for Realistic Animation of Clouds*[1]. **Fig. 1** shows a final result of this research.

First, the details of *Simulation Method* that is introduced by *Dobashi et al.* are described. Then *Rendering Method* of the simulation is discussed. The report is concluded by showing the final results of final-state of the implementation. Sections II, III, and IV, which explain the theory behind implementation, are adapted from [3].

II. Basic Idea

In the implementation, I followed the same approach that of *Dobashi et al.*'s. The method consists of two processes, simulation and rendering. As shown in **Fig. 2(a)**, the simulation space is a cellular automaton. At each cell, there are three *binary* properties; vapor/humidity (*hum*), clouds (*cld*), and phase transition (*act*). Cloud simulation is realized by applying some transition rules based on those properties at each frame.

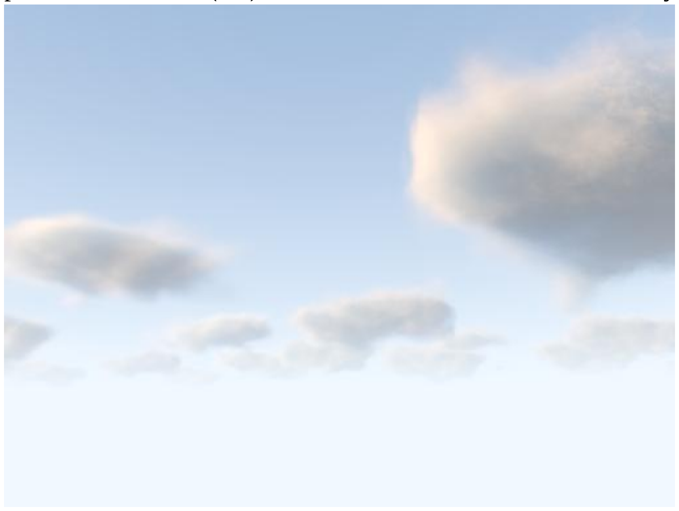


Fig. 1. A final result from the project.

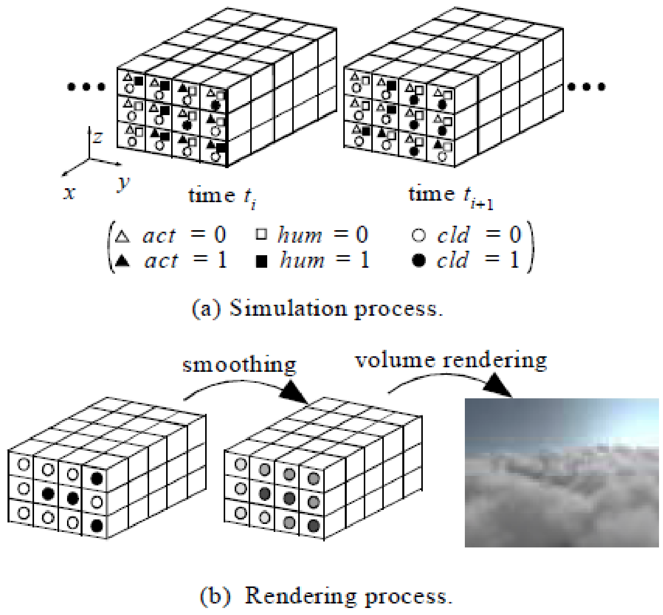


Fig. 2. An overview of *Dobashi et al.*'s method

What is obtained from this simulation is just binary representation of each cell either *having a cloud* or *not having a cloud*. Therefore, a smoothing operation is needed before the rendering process as shown in **Fig. 2(b).** After the smoothing is done the scene is rendered using volume rendering techniques.

III. Simulation Method

It can be said that simulation method has two level of hierarchy. The overall shapes of the clouds are controlled using ellipsoids and the dynamics like cloud particle generation and extinction are controlled using cellular automaton representation.

A. Growth Simulation

As mentioned at *Basic Idea*, the cells in the automaton have three variables, *hum*, *act* and *cld*. For simplicity, the simulation space is aligned with *xyz* axes. Each represents vapor, phase transition factor and cloud respectively. *hum=1* means that the vapor amount is enough to form cloud, *act=1* means the phase transition from vapor to water ready to occur and *cld=1* means the cell has cloud particle. With the light of these explanations, the transition rules are as follows.

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

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

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

where $f_{act}(i,j,k)$ is a binary function and its computed according to adjacent cells' *act* values. $f_{act}(i,j,k)$ is as follows.

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

Beginning from initial status, cloud growth is simulated using Eqs. (1) through (4).

B. Cloud Extinction

Problem with *Growth Simulation* transitions is when a cloud is formed at a cell it never extinct. So, an extinction algorithm is also needed for a realistic simulation. The proposed method by *Dobashi et al.* is as follows. First three probabilities are assigned to each cell; p_{ext}, p_{hum} and p_{act}. Then a random number, *rnd* ($0 \le rnd \le 1$) is generated, and *cld, hum* and *act* variables are changed according to this random number. The methods for those transitions are as follows.

$$hum(i,j,k,t_{i+1}) = hum(i,j,k,t_i) \vee is(rnd < p_{hum}(i,j,k,t_i))\ , \tag{5}$$

$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \wedge is(rnd > p_{ext}(i,j,k,t_i))\ , \tag{6}$$

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

The overall of shape of clouds can be adjusted by manipulating those probabilities accordingly. Some usage of this phenomenon is mentioned at *Controlling Cloud Motion Using Ellipsoids*.

C. Advection by Wind

The cloud motion under wind is another important deal of cloud simulation. To simulate it, simply a transition rule which shifts all *cld*, *hum* and *act* variables toward wind direction is enough. For simplicity, *Dobashi et al.* assumed wind always blows toward x-axis direction. Also since the wind speed varies at different heights, a velocity function $v(z_k)$, which depends on the z-coordinate of a cell (i,j,k) is proposed. The transition rules are as follows:

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

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

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

Unfortunately, advection by wind aspect of the simulation is not implemented at final project. But, as can be seen from those equations, it is really easy to add that feature into simulation method with a few lines of codes.

D. Fast Simulation Using Bit Field Manipulation Functions

Since each variable is a binary number, they can be represented by one bit each. So to reduce the computational overhead, instead of using *char* for each variable, by using *unsigned long* values, thirty-two cells can be grouped into one variable. Then by applying bitwise boolean functions, thirty-two cells can be processed at the same time.

In the final implementation of cloud simulation, fast simulation techniques that are discussed here are omitted for two reasons; the aimed simulation is not real-time, and even without the fast simulation techniques, the simulation step takes nearly no time, compared to rendering.

E. Controlling Cloud Motion Using Ellipsoids

As mentioned at *Cloud Extinction*, the animator can control the overall shape of clouds by manipulating probability values. Ellipsoids are good candidates for this control. They can be used to simulate air parcels which have a higher probability of cloud generation at their centers and cloud extinctions at their edges. This means having higher p_{hum} and p_{act} at ellipsoid's center than at its edges. Inversely, p_{ext} should be lower at the center than the edges.

The wind effect can also be simulated for those ellipsoids. By simply moving the ellipsoids position with respect to wind, while shifting the variables as described at *Advection by Wind*, a good wind simulation can be achieved.

IV. Rendering Method

The method which is followed for rendering is described at this section. First the cloud density distribution will be calculated using binary results of the simulation. Then, instead of implementing the method introduced by *Dobashi et al.*, calculated density grid will simply be fed to *PovRay* tool, which will handle the final volume rendering.

A. Continuous Density Distribution Calculation

The density distribution that comes from simulation results is binary, unlike the continuous case of real world. Therefore using *Gaussian Filtering* a smoothing should be performed. For each cell (i,j,k), corresponding density value is calculated for each time step t_i using following equation:

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

Where w is the Gaussian weight function and i_0, j_0, k_0, t_0 are the window sizes of this Gaussian. It should be noted that time variant is also included in the smoothing since the distribution of cloud particles with respect to time is also discrete. I have used Matlab for this 4D Gaussian filtering job. First a 4D

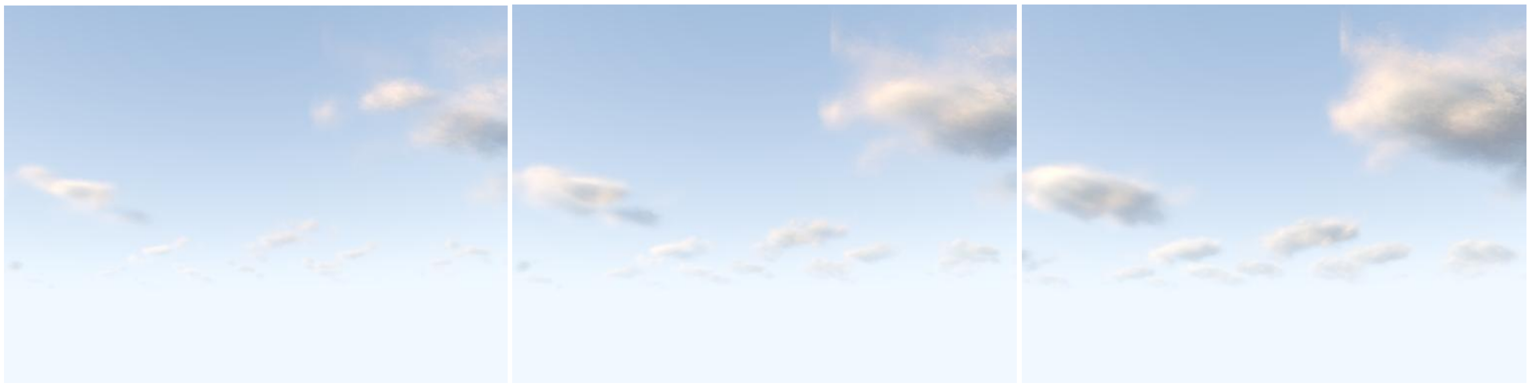

(a) t_i=10 (b) t_i=20 (c) t_i=30

Fig. 3. Final State of Implementation.
Both growth and extinction is occurring.
Particles are forming with respect to ellipsoid shapes.
Rendering is done using radiosity and scattering.

Gaussian Kernel with window size 5 is created. And then, this window is convolved over density distribution.

As far as I understand, those cells of the automaton could both represent a pixel or a set of pixels for the scene. In the latter case, an interpolation for actual pixel values will also be needed. In the implementation, 32x32x32x30 density distribution (the last term is time) is interpolated to a final volume of 125x125x125x291 using *interp* methods of Matlab.

B. *Volume Rendering using PovRay*

The Persistence of Vision Raytracer (*PovRay*) [4] is a high-quality; totally free tool for creating stunning three-dimensional graphics.

Although, another tool *PBRT* was initially proposed for rendering job at [3], but further investigation revealed *PovRay* is both more suitable and easy to learn tool for rendering process. So, after density distribution is calculated, *PovRay* is used to render the scene. The tricky part of this job was to learn how to set *PovRay* variables to end up with a realistic scene. Fortunately, throughout the research, I have found a sample configuration for cloud rendering at [5]. Using their scene with some modification, a realistic rendering of clouds is achieved as already seen at **Fig. 1**.

At *PovRay*, a scene with a simple sun and a sky sphere is set up first. Then, density data which is saved as *.df3* format after *Continuous Density Distribution Calculation* is rendered with *scattering*. *Radiosity* is enabled for this rendering process as well (see Appendix A, B).

V. Results

A. *Simulation*

All *simulation* details explained at this report is implemented; except for *Advection by Wind* and *Fast Simulation Using Bit Field Manipulation Functions*. The results of the simulation can be seen at **Fig. 3**.

One extension of *Dobashi et al.'*s work is about the probability distributions' of *cld*, *hum* and *act* variables. I want to achieve a complete simulation, starting from a cloud's initial creation to the extinction. Therefore, rather than having constant probabilities over time, I have changed them so that as time pass, the extinction probabilities increased, inversely the humidity probabilities dropped resulting in a complete extinction. This approach gave fairly good simulation of a cloud's life.

One difficulty that is faced during simulation implementation is about those probability values. To have a good simulation, they must be calibrated very carefully. After some experiments, I followed the default values proposed by *Dobashi et al.* So, p_{hum}, p_{act}, and p_{cld}, values are 0.1, 0.001 and 0.1 respectively, at the centers of ellipsoids.

The simulation step that is implemented runs fast. For a cellular automaton of size 32x32x32, and a time-step size of 30, the overall execution time is ~0 min.

B. *Smoothing*

Smoothing and expansion is implemented at Matlab. For smoothing, a 4D gaussian kernel is used. One difficulty of this

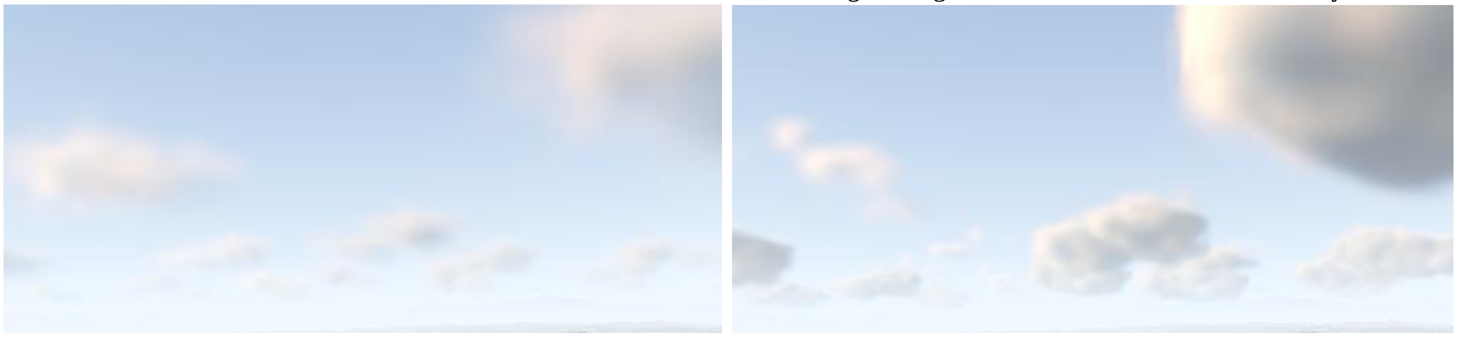


Fig. 4. Gaussian Kernel: Low Amplitude (0.03) vs. Normal Amplitude (1)

smoothing process is to adjust a proper amplitude value for the kernel. Initially, I was trying to adjust the amplitude (of 0.03) so that all values in the kernel window will sum up to 1. But this amplitude resulted in very hazy clouds, almost like fog (see **Fig. 4**). So, after some further experiments, I found that the ideal amplitude for gaussian filtering is 1.

Because of the enormity of the matrix (32x32x32x30), this smoothing process takes up to 1 hour on a PC with 1.83 GHz C2Duo Processor and 2Gb of Ram.

C. *Rendering*

As explained rendering is done using *PovRay*. In the *PovRay* scene that is used, overall radiosity as well with scattering for clouds is enabled. The scene is composed by multiple occurrences of the same volumetric data at different positions. To further increase the randomness, the initial and ending time-steps of those occurrences are also different than each other.

As an extension to those, also a turbulence function is applied top of density volume data. It is as easy as a one-line code in *PovRay*, yet it is really effective. Resulting final rendering after turbulence is applied can be seen at **Fig. 5**.

As can be guessed, the rendering process is the most time consuming step of cloud simulation. On the same PC with 1.83 GHz C2Duo Processor and 2Gb of Ram, rendering a sequence of 291 frames takes up to 4 hours.

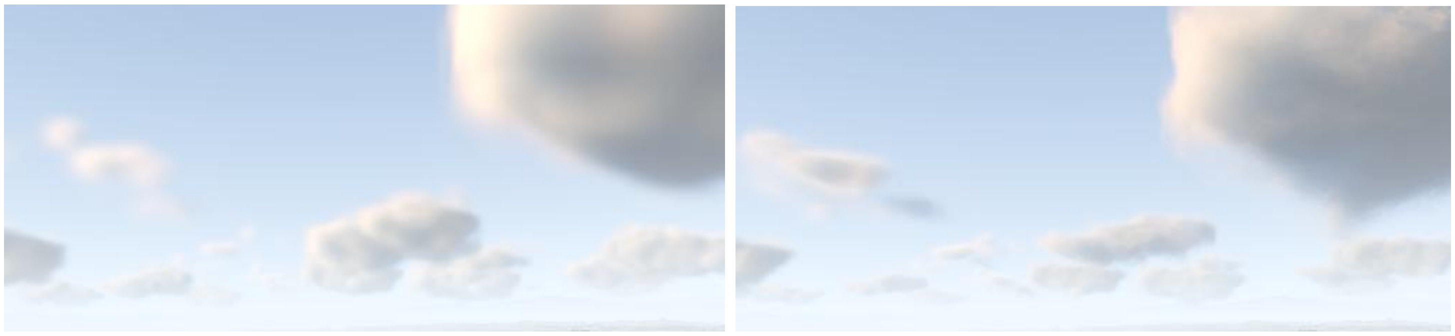


Fig. 5. Clouds with no turbulence vs. with turbulence.

VI. Conclusion

Final state of the implementation satisfies my desires. Though, I didn’t have a chance to implement an advection by wind behavior, the rendering of clouds is far better than my expectations. Also, I am very pleased with the ease of use of the implementation. I have created bunch of batch files, which helps user along the whole simulation process.

As I always do, I want to conclude the paper by showing some results of the work. Please find them below.

Appendix

A. *PovRay Radiosity*

```
global_settings {
    assumed_gamma 1
    max_trace_level 256
    #if (RadOK=1)
        radiosity {
            count 10
            recursion_limit 1
            low_error_factor .5
            gray_threshold 0.0
            minimum_reuse 0.015
            brightness 1
            adc_bailout 0.01/2
        }
    #end
}
```

B. *PovRay Cloud Object*

```
#declare Cloud1=box{-0.5,0.5
    texture{
        pigment{Clear}
        finish{ambient 0 diffuse 0}
    }
    hollow
    interior{
      media{
          scattering{1,C_Sun*0.006
                extinction 0.25}
            intervals Intervals
         density{
              density_file df3 "x.df3"
                   turbulence Turbulence
                   lambda 3
                   interpolate 1
                   translate -0.5
                   scale <1,-1,1>
            }
         }
      }
      scale <4/3,1,1>
    }
```

References

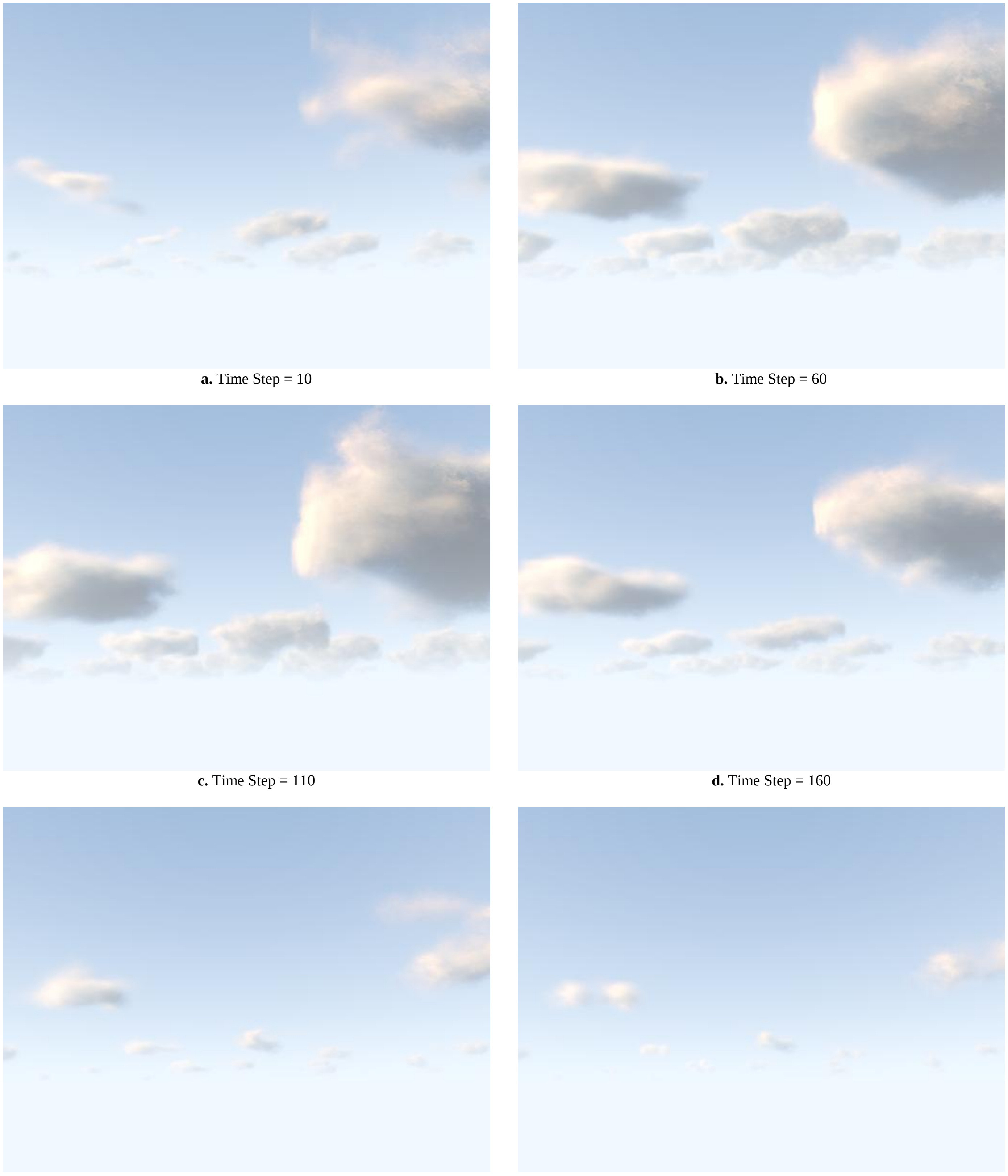
[1] Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T., and Nishita, T. 2000. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th Annual Conference on Computer Graphics and interactive Techniques* International Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 19-28.

[2] Pharr, M. and Humphreys, G. 2004 *Physically Based Rendering: from Theory to Implementation*. Morgan Kaufmann Publishers Inc.

[3] Yildiz, C., A Progress Report on a Simple, Efficient Method for Realistic Animation of Clouds.

[4] The Persistence of Vision Raytracer, http://www.povray.org/

[5] Oyonale - 3D art and graphic experiments, Cloud (POV-Ray) http://www.oyonale.com/modeles.php?page=36



a. Time Step = 10

b. Time Step = 60

c. Time Step = 110

d. Time Step = 160

e. Time Step = 210

f. Time Step = 260

Fig. 6. Life span of a scene with clouds.