Real Time Animation of Realistic Fog

Venceslas Biri, Sylvain Michelin

Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Real-Time Animation of Realistic Fog

V. Biri, S. Michelin and D. Arquès

University of Marne-la-Vallée, Gaspard Monge Institute, France

Abstract

Fog introduces a high level of realism to computer graphics. But fog is often simulated by uniform density when real fog is always much more complex. Its variable density creates beautiful shapes of mist what can add a realistic ambience to virtual scene. We present here a new algorithm to render such complex medium in real-time and propose a mean to design non homogeneous fog using chosen functions. Then we take advantage of fog properties and of graphic hardware to achieve fast rendering. We also present a method to integrate wind effects and fog animation without expensive cost in time.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Display algorithms

1. Introduction

Integrating participating media in image rendering remains a real challenge. But such images are important or even necessary for several applications[13], including safety analyses for smoke, military and industrial simulations, entertainment, cinema, driving and flying simulators... As a special but current participating medium, fog adds to images a very realistic effect. Simple model of fog is for example very used in real-time to enhance realism and provide important depth culling.

Two main ways have been proposed to simulate fog. The first one considers it as a standard participating medium, and rely on physical equations[16] to solve the illumination problem induced by such medium. Both determinist and stochastic attempts are used in this way. Determinist methods group extensions of radiosity algorithm, like the zonal method[14] and other improvements[17, 18], with algorithms which use spherical harmonics or discrete ordinates[1, 6, 9]. In the same way, other determinist methods use implicit representations of directional distribution of light[10, 19]. Stochastic methods use random sampling to render such medium, using Monte Carlo techniques[2, 5, 11] or more recently photon maps[4]. A detailed overview of most previous methods can be found in the Pérez et al.[12]. All these methods produce very realistic images of participating medium but they suffer of a long computation time especially when we have to deal with fog in outdoor scene recovering the whole image.

A second way takes advantage of the fog properties to make several approximations allowing fast rendering. A standard and simplistic model, which considers a uniformly dense fog, is used in real-time graphic APIs, such as OpenGL[15]. But further complexity can be achieved to simulate light sources effects[7] or design height dependent density[8, 3]. These two last methods attempt to slightly render more complex fog and focuses on linear method relying only on graphic card capacities. All these methods allow simple but fast representations of fog.

In this paper, we propose a new real-time rendering algorithm allowing complex representation of fog but without global illumination determination. We present a new approach to build user-defined shapes of mist using a set of functions. Moreover, the function decomposition allows also to add wind effect, and to animate such complex medium.

In the next section, we review the transport equation governing illumination of participating medium. In section 3 we focus on fog properties to achieve real-time before presenting the algorithm in section 4. Section 5 introduces wind effects and more complex fog animations. Finally results are shown in section 6 before concluding in section 7.

2. Theoretical background

Fog is a cloud of little water droplets in suspension. When an emission of light enters such medium, the light is scattered through the cloud. Different amounts of light depending on its density are then captured, reflected and emitted in

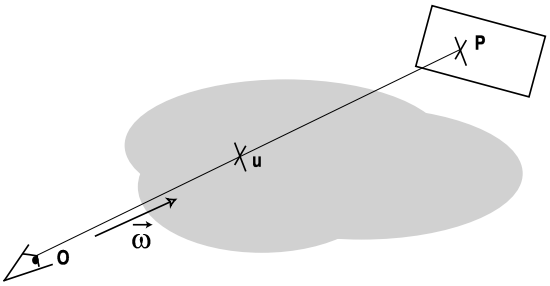

Figure 1: *Ray of incoming light in O from P through a participating medium.*

all directions. In computer graphics, fog belongs to participating media. As light travels along a ray of direction $\vec{\omega}$ in such medium illustrated in figure 1, four interactions[16] modify the radiance L : absorption, emission, scattering and in-scattering.

Absorption is mainly responsible for the loss of light. It represents transformations of radiant energy along the ray in another energy form, inducing a reduction of radiance. Emission is the opposite process, creating spontaneous light along the ray. This is, for example, what happens in neon lighting. Scattering and in-scattering are the results of deviations in light direction induced by the medium. Scattering occurs when the incoming light is deflected from $\vec{\omega}$ in other directions, and in-scattering when light, after being deflected in the medium, follows $\vec{\omega}$. In-scattering produces a raise of radiance along the light ray because light incoming from other directions are reflected in the medium to the direction $\vec{\omega}$.

Absorption and scattering have similar effects and we can group both phenomena in the same equation :

$$\frac{dL(u,\vec{\omega})}{du} = -K_t(u)L(u,\vec{\omega}) \tag{1}$$

where K_t is the extinction coefficient defined by :

$$K_t(u) = K_s(u) + K_a(u)$$

K_a is the absorption coefficient and K_s the diffusion coefficient. Emission involves an augmentation of radiant energy :

$$\frac{dL(u,\vec{\omega})}{du} = K_a(u)L_a(u) \tag{2}$$

being L_a the emissive light energy.
The incoming scattered light from in-scattering is modeled by a phase function $p(\vec{\omega}_i,\vec{\omega})$ expressing ratio of light coming from any direction $\vec{\omega}_i$ which follows direction $\vec{\omega}$.

$$\frac{dL(u,\vec{\omega})}{du} = K_s(u)\int_{S^2} L_i(u,\vec{\omega})p(\vec{\omega}_i,\vec{\omega})\,d\vec{\omega}_i \tag{3}$$

being S^2 the entire sphere surrounding the considered point.

Addition of equations (1), (2) and (3) gives the transport equation[16] expressing the radiance variation created by the four interactions in point u and direction $\vec{\omega}$:

$$\frac{dL(u,\vec{\omega})}{du} = -K_t(u)L(u,\vec{\omega}) + K_t(u)J(u,\vec{\omega})$$

where $J(u,\vec{\omega})$ is the source radiance :

$$J(u,\vec{\omega}) = \frac{K_a(u)}{K_t(u)}L_a(u) + \frac{K_s(u)}{K_t(u)}\int_{S^2} L_i(u,\vec{\omega})p(\vec{\omega}_i,\vec{\omega})\,d\vec{\omega}_i$$

It takes into account the radiance added in point u in the direction $\vec{\omega}$ due to self-emission and in-scattering. Solution of previous equation is the integral transport equation from a point O to a point P in the ray :

$$L(O) = \tau(O,P)L(P) + \int_O^P \tau(O,u)K_t(u)J(u,\vec{\omega})du \tag{4}$$

where τ is the transmittance along the ray :

$$\tau(u,v) = \exp^{-\int_u^v K_t(x)dx}$$

The first term of equation (4) expresses the reduced light coming from point P (if any). The second term represents light added along the ray from point O to point P by self-emission and in- scattering.

3. Fog and real-time

Our objective is to render realistic fog, in real-time. Since fog is a special participating medium, several simplifications could be done to improve calculations. In this section, we show how time saving can be used to simulate more complex fog through the use of chosen functions.

3.1. Fog approximation

If we consider a outdoor scene in daylight, fog will have mainly two effects : a drain of any color received by the eyes and a creation of a veil of white mist. Since daylight is very scattered in a fog, we could consider that in-scattering can be represented by a constant amount of light L_{fog} and that emission can be neglected. Then, source radiance $J(u,\vec{\omega})$ is constant and equal to L_{fog}. Equation (4) becomes :

$$\begin{aligned} L(O) &= \tau(O,P)L(P) + L_{fog}\int_O^P \tau(O,u)K_t(u)du \\ &= \tau(O,P)L(P) + L_{fog}(1-\tau(O,P)) \end{aligned}$$

In that case, each pixel with color C_{in} drawn in the image is blended with the color of the fog C_{fog} to obtain the final color C_{fin} :

$$C_{fin} = fC_{in} + (1-f)C_{fog} \tag{5}$$

where coefficient f is :

$$f = \tau(O,P) = \exp\left(-\int_O^P K_t(x)\,dx\right) \tag{6}$$

In equation (5), the first term represents the loss of light

from the incoming color due to for scattering and absorption of light while the second term is responsible for the drain of color and the white veil simulating the important in-scattering of the fog. In classical rendering APIs such as OpenGL, this fog approximation is generally taken into account but with a constant fog density and an approximated distance OP. This induces a very simple representation of fog.

3.2. Defining complex fog

But in real life, fog is always a complex combination of layers and shapes. This is specially visible when we travel quickly, driving or flying, in the fog or when wind changes its shape. Variations in shape and density could be represented by variations of the extinction coefficient, which becomes in fact an extinction function.

The main idea is to decompose this function into a set of functions, i.e. it equals a weighted sum of N chosen functions γ_i.

$$K_t(x) = \sum_{i=1}^{N} c_i \, \gamma_i(x) \tag{7}$$

Since we have to integrate them, functions γ_i must be chosen carefully. Indeed, we want to know the analytic integrale Γ_i of these functions γ_i to avoid a numerical computation of the transmittance along the ray. Then the equation (6) becomes :

$$\tau(O,P) = \exp\left(-\sum_{i=1}^{N} c_i(\Gamma_i(P) - \Gamma_i(O))\right) \tag{8}$$

Functions Γ_i has to be defined over the entire scene. To avoid coarse function without precision, a good choice is to take periodic functions that can be precise and defined everywhere.

In the following section, we present some function families we choose for their simplicity or the convenience they provide. Although others could be chosen, such family can easily define generic complex fog.

3.2.1. Cosine functions

N cosine functions can be used to represent the extinction coefficient :

$$K_t(x) = c_0 + \sum_{i=1}^{N-1} c_i \cos(k_i \Lambda_i(x) + \phi_i)$$

Λ_i are special operators on 3D point x like projection on one axis or on a chosen direction. Coefficients c_i, k_i, ϕ_i and operators Λ_i are chosen by user. Nevertheless extinction function must not be negative, in taking for example :

$$c_0 \geq \sum_{i=1}^{N} \|c_i\|$$

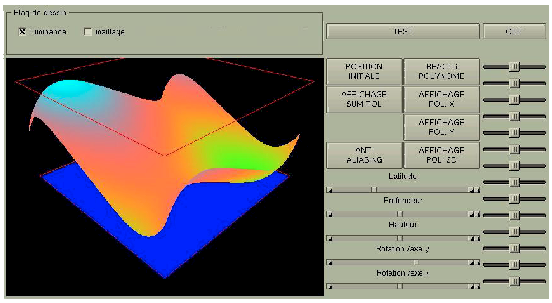

Figure 2: *Partial view of the application used to define fog density with polynomials*

These functions have been chosen because they are naturally periodic and can easily introduce phase difference. Yet, operators Λ_i must be very simple to allow an analytic integration. For example, we used only projection on elementary axis or fixed direction.

3.2.2. Polynomial functions

N polynomial functions can also be used to represent the extinction function :

$$K_t(x) = \sum_{i=1}^{N} c_i P_i(x)$$

P_i are polynomial functions on 3d point x and c_i are chosen coefficients. As polynomials could be easily integrated and combined, we can use one, two or three dimensional polynomials. Of course, if they are chosen with two or three variables (dimensions), it will increase computational time. Nevertheless, since they are simple functions, polynomials allow quick computations. And they also give the possibility to choose easily and intuitively the 3d shape of fog as shown in the figure 2 which represent a user-defined distribution of fog density. So defining a shape for the fog density is really easy and intuitive. Once again, attention should be paid to build positive function.

Unfortunately, polynomial functions are not periodic. So we have to make them periodic in restricting their interval of definition to some bounded area and duplicating this area all over the scene, as illustrated in figure 3 for a 2 dimensional polynomial. These polynomial can be also considered null outside their area of definition to obtain more precice fog shape. In the examples of section 5, we use 1D polynomials defined on intervals $[-C \ldots C]$ where C is a constant set by the user to fit the scene. The larger this coefficient is, the wider and coarser the fog will be and the quicker the algorithm. The smaller C is, the finer fog will be and the slower the algorithm.

When we integrate the extinction function along a path we just have to decompose each portion of the path in each bounded area and then integrate these portions translated in

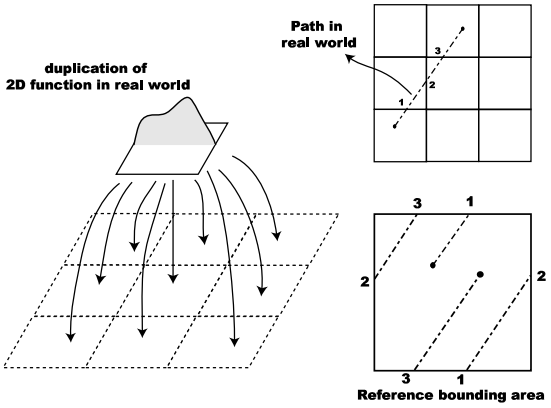

Figure 3: *Duplication of a non periodic function and integration along a path for such function.*

the reference area as illustrated in figure 3. This virtual periodicity induces to choose carefully coefficients to obtain the same value in the border of the reference bounding area.

3.2.3. Equipotential functions and combinations

To add precise and complex shapes of the fog, we also introduce equipotential functions. Instead of integrating along a ray, we suppose that extinction coefficient depends on a potential created by a defined virtual object : point, line or sphere. Adding and subtracting such virtual objects leads to complex potential functions and, as a consequence, sophisticated fog shapes.

The potential functions we use depend on the distance between the considered ray and the virtual object. For example, with points and lines, we use the function :

$$p = \frac{1}{c + dr^2} \tag{9}$$

where r is the distance between the segment OP and the virtual object, c and d two chosen parameters.

The main interest using such functions is to blend them with the functions described above. Indeed, the integration being linear, any functions of previous families can be combined to obtain complex fog defined everywhere but with local precise behaviors.

4. Algorithm

We present in this section the algorithm to render previous complex fog in real-time taking profit of graphic hardware.

4.1. Fog rendering

For each pixel of the screen, fog must be rendered using equation (5) which mixes incoming color (i.e. the color of point P) with the fog color C_{fog}. Coefficient f, representing the transparency, is computed with equation (8). To obtain this blend we draw a plan, which has the fog color, in the screen position and put a transparency texture on it as represented in figure 4. The transparency is used by OpenGL hardware to blend incoming color with the fog color.

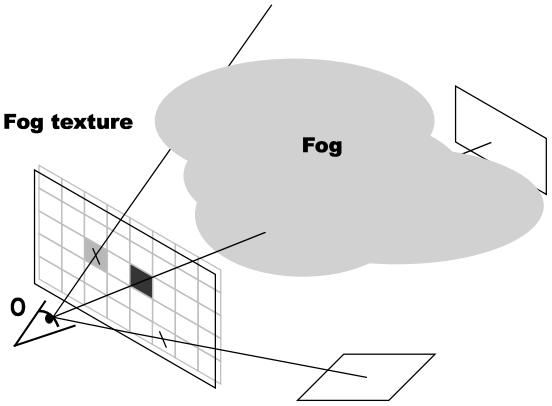

Figure 4: *Simulation of fog by drawing a plan with a fog texture.*

We take advantage of texture mapping to fit texture size of fog with the size of the screen. Since we could choose the size of the fog texture, we are able to maintain real-time, in choosing a smaller texture when it is needed. Drawback is that the smaller the texture is, the larger the aliasing will be.

4.2. Computation of transparency

For each pixel of the fog texture, we must integrate the extinction function along the ray between the eye and the point represented on the screen at this position. To compute this integral, eye coordinates and viewed point coordinates are needed. By using the depth buffer of graphic cards, we obtain the depth of the viewed point. Combining this depth with viewed point position on the screen, and multiplying them by the inverse of the projection matrix, gives the viewed point coordinates in the real scene.

If depth is beyond a viewing threshold, no more calculations are needed and the transparency will be set to 1 to represent the fog color. It means that far points are not seen in case of fog what allows also depth culling and speeds the algorithm. For all other points, integral is resolved by adding all contributions of the N functions used to describe the extinction function using equation (8).

4.3. Antialiasing and buffer method

The main drawback of this method is an aliasing effect, especially when texture size is small compared to image size. One cheap way to reduced that effect, is to scale the depth

map obtained by the graphic cards in the size of the fog texture. When scaling, each final depth point is then averaged with its neighbor pixel depths. Of course this work has to be done by graphic card if possible otherwise it would be too slow.

To avoid totally aliasing, the fog texture must have a greater size than the image, but in this case computation time will increase. So we design a buffer method borrowed from [3] to quickly compute the fog texture taking advantage of graphic hardware. Instead of computing each texture pixel, a first pass consists in projecting in a auxiliary buffer each patch of the scene, with red vertex color equals to the transparency computed as defined in section 4.2. We can use the front buffer if auxiliary buffer is not enabled. Then, this buffer will be used as the fog texture. In this case, we don't have to read the depth buffer and to compute each viewed point positions since we use already known vertex postions.

Despite the linear interpolation, assumed by graphic hardware, of the transparency between each point used to define objects in the scene, large textures are computed quickly and aliasing is avoided. In this method, computational time depends above all on number of vertices and not on the texture size contrary to the previous method.

4.4. Algorithm overview

```
Choose size of fog texture
For each frame
  Display the scene
  Recover the depth map
  If (antialiasing is set)
    scale the depth map to texture size
  Recover the inverse of matrix projection
  Recover the eye position
  Compute fog for eye position
  For each pixel of fog texture
    Recover its screen position
    Recover its depth
    Multiply screen coordinate by projection
      matrix
    Recover distance between eye and point
    Compute f using equation 6 and 8
  End_for
  Display texture
End_for
```

5. Wind effect and animation of fog

With this representation of fog, we could integrate easily a realistic effect of wind. Each function will be tagged to be sensible or not to the effect of time and wind.

First, we set a direction of wind propagation (which could change in time) and a wind speed. The extinction coefficient (i.e. each parameter of functions used) are linked to these sliding factors.

The effects of this sliding on cosines functions is easily integrated since they are periodic. Sliding factors are just added to coefficients ϕ and the rendering of fog is done in the same way as before. For polynomial functions, this is slightly more complicated since these functions are not naturally periodic. But the effects of sliding could be computed inexpensively by sliding the start and end point of integration in the reference bounding area as illustrated in figure 5. For equipotential function, only the pattern positions have to be changed.

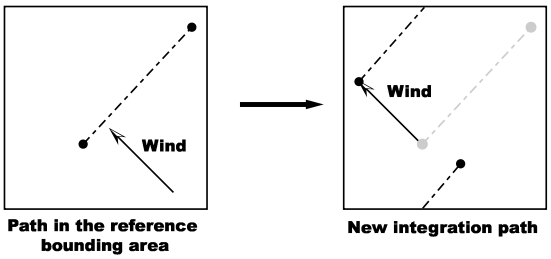

Figure 5: *Integration of wind effect on polynomial functions.*

But other effects could be achieve using time dependent functions. Instead of just translating the functions, any change of function parameters introduces animations of the inner shape of the fog. In the cosine function base, changing coefficients c will modify importance of the functions between themself. Any modifications of coefficients k will shrink or expand the function. For polynomial functions, changing parameters disturbs completely the fog shape. If we designed two polynomial shapes of fog, a linear interpolation between the coefficients of both polynomials will lead to a similar transformation of the resulting fog. In the case of equipotential functions, both parameters c and d (cf. equation 9) and positions of virtual objects could be modified.

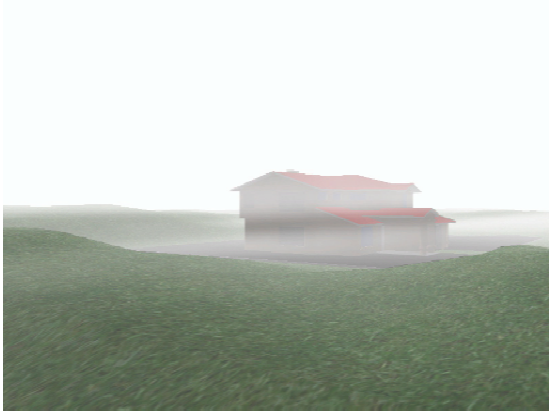

Figure 6: *Fog defined with cosine functions*

6. Results

An implementation of this algorithm has been designed using QT and OpenGL on a AMD 1600 processor associated with a GeForce 2 graphic card. Lighting is done by OpenGL with a directional light simulating the daylight (global direction of the sun). To render the multiscattering, a high ambiant term has been chosen. Rendering is done without shadow casting but a Z-buffer method can be introduce quickly since we already read the depth buffer and so it will not slow down our algorithm. We don't have introduce it because, in case of fog, there is few shadow effects. The images of figure 6 and 7 are 800x600 pixel wide and use a 512x512 fog texture. They show different views of a contryside (about 21200 triangles) in daylight (directional light) with complex mist.

The figure 6 present complex and realistic layers of mist. The extension function is a combination of four cosine functions plus one constant.

$$K_t(x,y,z) = 1 + \frac{1}{2}cos(5\pi y) + \frac{1}{5}cos(7\pi(y+0,1x)) + \frac{1}{5}cos(5\pi(y-0,05x)) + \frac{1}{10}cos(\pi x)cos(\frac{\pi z}{2})$$

Figure 7: *Fog defined with polynomial functions. Two waves of density can be seen in the right and in the left.*

The image of figure 7 presents a fog defined with a sum of two simple polynomials of fifth degree with coefficients $\{0.8,-1,0,0,1\}$ on the two horizontal axes X and Z. We point out variations of the shape in this image. This image shows that, with polynomials, it is easy to choose which part of the scene will be in the fog. In that extrem example, we can see the middle of the house but the tree and the house extremeties are surrounded by the mist.

Figure 8 shows the use of equipotential functions. Here again the texture have been chosen to 512x512. Four point equipotential functions, in the first image, are centered in the trees to enhance the mist in this area. The effect produced is that fog seems to concentrate on the threes.

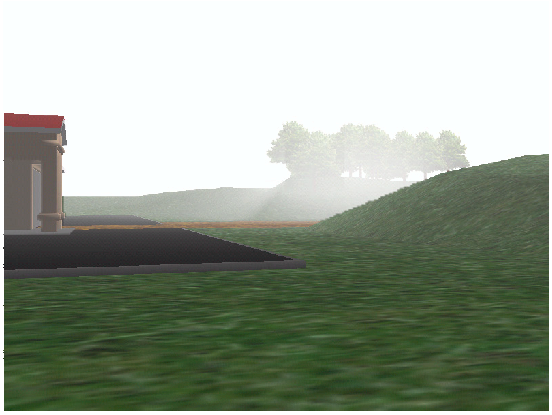

Figure 8: *Constant fog with equipotential point functions.*

Table 1 shows the number of frame per second we obtain for different fog texture resolution in the first outdoor scene (figure 6). We compare two differents functions : polynomials defined above for this scene and a constant function. Image size is always 800x600 and without fog, we obtain almost 50 frame per second. In brackets are indicated the frame per second for the antialiasing algorithm of section 4.3. We can see that the antialiasing algorithm is more sensitive to the scene complexity than to the texture size. So for large texture, it provides a great speed up in time. Moreover, in this case, we doen't use any optimisation to reduce the number of vertices drawn (all the scene is drawn) what would greatly speed up time presented for the antialiasing method.

Resolution	512x512	512x256	256x256	256x128	128x128
Polynomials	3.5 (7)	6.7 (9)	12.5 (11)	20 (12)	30 (12.5)
Constant	6.6 (11)	11.5	20 (23)	30	40 (33)

Table 1: *Frame per second for scene of figure 6*

Figure 9 point out aliasing produced by fog texture, comparing resolution 512x512, 256x256 and 128x128. For resolution 512x512, aliasing disapears. And finally, figure 10 shows four images from an animation illustrating wind effect on fog. These images are 800x600 pixel wide and use a 256x256 texture size. The scene contains 250000 triangles but our algorithm still maintains its performances (about 20 fps for resolution 128x128). Wind is sensitive when we see the shape of fog passing through the house and the bridge.

Color pictures and animations can also be found in http://www-igm.univ-mlv.fr/~biri/index.html