Modeling and rendering heterogeneous fog in real-time using B-Spline wavelets

Anthony Giroud
University Paris Est Marne-la-Vallée
giroud@univ-umlv.fr

Venceslas Biri
University Paris Est Marne-la-Vallée
biri@univ-umlv.fr

ABSTRACT

Heterogeneous fogs are often modeled with several layers of different density or using particle systems. However, layers are limited to vertical variations and using particles can involve a long computation time with large outdoor scenes. In this article we present a simple method to render heterogeneous fog in real-time. The extinction function of our fog, related to its density, is first modeled in a B-Spline function basis. Then, a wavelet transform is applied on this function to obtain a decomposition in both space and frequency domains. A grid traversal is used to render the fog in real time using the GPU. Since no precomputation is required concerning the position of the camera or the fog, we can freely navigate or move the fog into the scene.

Keywords: Participating medium, Fog, Rendering, GPU.

1 INTRODUCTION

Fog is massively used in rendering both for aesthetic purposes and to increase performances by providing an efficient way to cull surfaces that are far from the camera. Simple fog models are straightforward to implement but, like OpenGL's fog model, only allow a basic representation of homogeneous fog as can be seen on figure 1. Most of the time, these models are barely convincing visually, as we know that natural fogs never reach such perfect homogeneity. Considering latest advances in GPU programming, design of heterogeneous fog should be simple, and its rendering reachable in real-time.

The fog phenomenon is due to small particles of water in suspension. Because it interacts with light rays, fog is considered as a participating medium in computer graphics. Fog effects take into account attenuation, caused by absorption and out-scattering, and also consider multiple scattering of light as isotropic and constant over the scene. If we consider an homogeneous fog in its simplest form, equations are simple enough to allow an analytical integration of its effects along a view ray. When rendering heterogeneous fog, the density of water particles is varying across the scene, thus dramatically complexifying the model, involving local changes in physical properties of the fog, such as its extinction coefficient. Therefore, in order to compute the light-fog interaction, we have no other choice than performing the integration of the density along each view ray from the eye to the nearest object.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

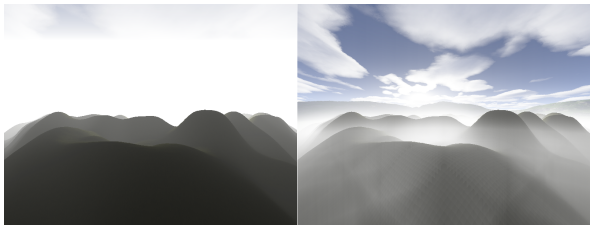

Figure 1: Comparison between OpenGL's homogeneous fog (left) and our heterogeneous fog (right)

Considerable work has been achieved in the development of real time solutions to handle participating media. Physical simulations taken aside [12, 14, 7, 11, 6], which do not reach realtime, researchers have been working on rendering complex exchanges of light within the medium, dealing, for example, with single scattering. They also considered simpler forms of fogs, with a density either varying along horizontal layers, defined by Perlin noise, or using particles. But few tried a direct and continuous mathematical representation of its density.

In this paper, we present a new method helping to shape and render complex heterogeneous fog in large outdoor scenes, lighted by a single light source (the sun). First, the fog is modeled in a B-Spline function basis, which allows a simple and efficient construction of its extinction function. As a preparation before rendering, Mallat's wavelet decomposition is applied on the extinction function in order to automatically generate different resolutions, enabling an optimized real-time rendering using the GPU. The use of wavelets offers several advantages :

- An easy modelization leading to a smooth and continuous fog density by opposition to particles ap-

proaches that are discrete. Analytical representation compresses data more efficiently and are, for example, easier to animate.
- Wavelet modelization is generic. It includes naturally, using Haar wavelets, discrete approaches like quad tree or octree representation.
- Wavelet decomposition leads to sparse data that can be used to improve rendering time.

Therefore the contribution of this paper is :
- Establishing a wavelet framework for the definition and modelization of an heterogeneous fog.
- Rendering the fog in real time using this representation without precomputation involving camera or fog position.
- Allowing a tradeoff between correctness and speed using the multiresolution offered by the wavelet decomposition.

In the next section, we review previous methods to render, in real time, the effects of participating media in a scene. Then, we briefly introduce the wavelet theory along with the equation of transfer inside a participating medium. Section 4 presents our modeling scheme and our implementation for rendering. In section 5, we expose and discuss our results.

2 PREVIOUS WORK

Rendering participating media such as fog in real-time has been well studied. We will not consider global illumination algorithms concerning participating media. For more information on this subject, the readers should refer to the excellent survey of Cerezo et al. [2]. Algorithms dealing with single scattering, including volume based approaches [17] or direct representation [1], also handle fog naturally but due to complexity problems these techniques only consider homogeneous mediums (except [19] discussed bellow). Therefore, we limit our overview to other real time approaches for heterogeneous fog which can roughly be divided in, on the one hand, particle approaches and, on the other hand, layered or bounded approaches.

Particles provide a natural way to handle heterogeneous fog. They have been used efficiently in numerous works [4, 15, 9, 3]. The idea is to consider particles as groups of water drops, allowing real time rendering of effects like smoke or physically based simulation. But is not well adapted to large scale fog recovering a whole scene. Moreover, animating all particles in a large scene is computer time consuming. The same drawbacks hold for the hybrid approach of Zhou et al. [19] which handles single scattering in a heterogeneous participating medium combining particles and spherical harmonics. We can also cite the work of Zdrojewska [18] which uses Perlin noise to alter the homogeneous density of the fog. Despite this good idea, the use of 3D random noise forbids any animation of this fog.

The idea behind layered or bounded approaches is to enclose fog density variations into layers [8, 5] or bounded volumes [10]. These works consider homogeneous fog enclosed in volume, inducing a discontinuous density function and creating artifacts on the border of these volumes. Moreover, intuitive or physically based animations of this kind of representation could be difficult to handle. Despite these limitations, it is often the kind of solution we can find in common graphic engines, along with particle rendering. Nevertheless, none of the previous methods offers a simple and efficient mathematical representation of heterogeneous fog adapted for both animation and rendering.

3 THEORETICAL BACKGROUND

3.1 Fog's illumination model

Our main goal is to render our fog in real-time, using conventional graphics cards. Although performances of GPUs have never been increasing so fast, we have to slightly simplify our fog model. Between points O and P, fog induces an attenuation (due to out-scattering and absorption) of the luminance L of P and an increase (in-scattering and emission) of light along the ray $\vec{OP}$. We start directly with the integral transfer equation, see [13] :

$$L(O) = \tau(O,P)L(P) + \int_O^P \tau(O,u)K_t(u)J(u,\vec{\omega})du \quad (1)$$

$L(O)$ being the radiance received by the observer, $J(u,\vec{\omega})$ being the incoming radiance along the ray, K_t the extinction coefficient and $\tau(u,v)$ the transmittance of the fog along the ray going from u to v :

$$\tau(u,v) = e^{-\int_u^v K_t(s)ds} \quad (2)$$

First, when daylight passes through fog, it is immediately scattered such that light in-scattering can be simplified by a constant amount L_{fog}. Moreover, if we consider that the light emitted by the fog itself can be neglected, the incoming radiance $J(u,\vec{\omega})$ equals L_{fog}, and then equation (1) becomes :

$$L(O) = \tau(O,P)L(P) + \int_O^P \tau(O,u)K_t(u)L_{\text{fog}}du \quad (3)$$

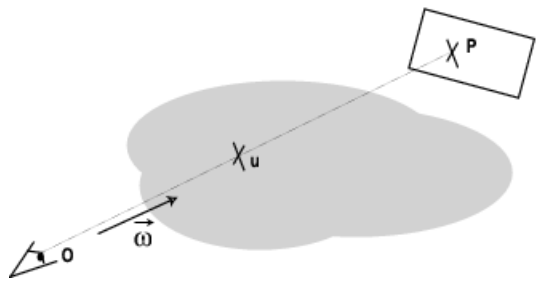

Figure 2: View ray $\vec{OP}$ through a participating medium.

The second part of equation (3) can be analytically integrated to obtain :

$$L(O) = \tau(O,P)L(P) + L_{\text{fog}}(1-\tau(O,P)) \quad (4)$$

3.2 Wavelets

From equation (3), we can see that the density variation could be represented efficiently by the extinction function. Therefore, K_t will be modeled using the wavelet framework whom principal used characteristics are detailed in this section. More details on the wavelet framework can be found in [16].

The wavelet framework. In a multiresolution analysis, data is represented using several approximation spaces. Different functions bases are used to represent a single signal, and each functions basis corresponds to a different resolution. Moreover, all basis functions are obtained by translating and scaling a single original pattern function $f \in \mathbb{L}^2(\mathbb{R})$, in other words :

$$f_{j,k}(x) = f(2^j x - k), \text{ with } j \in \mathbb{N}, k \in \mathbb{Z} \quad (5)$$

where $f_{j,k}$ represents the basis functions and j the resolution level. If we define F_j as the closed subspace of $\mathbb{L}^2$ using basis functions $\{f_{j,k}\}_{k\in\mathbb{N}}$, the closure of $\bigcup_{j\in\mathbb{N}} F_j$ is the space $\mathbb{L}^2$ and represent all square integrable functions.

The wavelet framework uses, to build basis functions of spaces F_j, a particular function called scaling function and often denoted by ϕ. It verifies equation (5) and generates a ϕ_{jk} family, $j \in \mathbb{N}, k \in \mathbb{Z}$. This function ϕ also presents the property of being written as a linear combination of $k/2$ translated and $1/2$ scaled versions of itself. It is the *scaling relation* of the scaling function, given by :

$$\phi(x) = \sum_{k=-\infty}^{\infty} p_k \times \phi(2x-k) \quad (6)$$

where $\{p_k\}$ are the coefficients of the *scaling sequence* of ϕ. Note that each subspace $F_j, j \in \mathbb{N}$ will in fact use the same and unique function ϕ translated and scaled.

The particularity of the wavelet framework is its ability to decompose a function of F_{j+1} using several functions of F_j and of its orthogonal complement G_j. Therefore, if J is the maximum resolution level, the F_J space can be written :

$$F_J = F_0 \cup \bigcup_{j=0}^{J-1} G_j \quad (7)$$

This equation means that a function (up to a resolution J) can be described using only one scaling function (space F_0) and several functions of spaces G_j. The basis functions of spaces G_j are called wavelet function and verify equation (5). They can also be built using the *scaling relation for wavelets*, which we will call the *wavelet relation* :

$$\psi(x) = \sum_{k=-\infty}^{\infty} q_k \times \phi(2x-k) \quad (8)$$

where $\{q_k\}$ are the coefficients of the *wavelet sequence* of ψ. Note that, similarly to F_j, each subspace G_j uses the same and unique function ψ translated and scaled.

Decomposition and multiresolution using wavelets. The advantage of the wavelet framework is that it provides an efficient way to decompose a function into multiresolution spaces. The fast decomposition can be assured by the Mallat's wavelet transform which uses, as entry data, coefficients of the function modeled directly in the maximum resolution level. Therefore, our fog extinction function will be modeled using scaling function.

Mallat's algorithm takes advantage of equation (7) and consists, for each step, in extracting from the approximation at level n (represented in a scaling functions basis) first the approximation at level $n-1$ (F_{n-1} which is twice less precise), and then the corresponding layer of details (G_{n-1} represented by a wavelet basis). We simply repeat this process until we obtain the approximation at level 0. Mallat's transform is lossless, therefore when we simply sum up the coarsest approximation with all layers of details, we recover the original signal untouched.

Wavelets in two dimensions. Now that we know how to build scaling functions and wavelets in one dimension, going 2D will actually be quite straightforward. In a nutshell, it simply consists in assigning the corresponding 1D function to each axis, and the result is given by the product of these two 1D functions. Basically :

$$\phi\phi(x,y) = \phi(x)\phi(y) \quad (9)$$

where $\phi\phi$ is a 2D scaling function and ϕ is the corresponding 1D scaling function. Things go exactly the same way with wavelet functions.

Obtaining a 2D wavelet transform is slightly harder and requires to process rows and columns separately. There are two different decomposition methods : the *standard decomposition* and the *nonstandard decomposition*. These two types of decomposition output exactly the same kind of result :

- A single coarse approximation at level 0, modeled with 2D scaling functions $\phi\phi(x,y) = \phi(x)\phi(y)$.
- $J-1$ layers of vertical details, modeled with hybrid functions $\phi\psi(x,y) = \phi(x)\psi(y)$.
- $J-1$ layers of horizontal details, modeled with hybrid functions $\psi\phi(x,y) = \psi(x)\phi(y)$.
- $J-1$ layers of 2D details, modeled with 2D wavelets $\psi\psi(x,y) = \psi(x)\psi(y)$.

For example, our fog's extinction function can be written as :

$$K_t = \sum_{ij} \alpha_{ij} \phi \phi_{ij} + \sum_{n=1}^{J-1} \left[\sum_{ij} \beta_{ij}^n \phi \psi_{ij}^n + \delta_{ij}^n \psi \phi_{ij}^n + \gamma_{ij}^n \psi \psi_{ij}^n \right] \quad (10)$$

4 OUR METHOD

4.1 Modeling the fog

The two-dimensional framework. Unlike other types of participating media from the same family, fog almost always appears in large outdoor scenes as a horizontal layer of varying thickness. This is quite different from smoke, which can evolve indifferently in all directions in terms of shape and movement, and thus really need to be defined with the same precision along all three dimensions.

For this reason, and in order to ease the shape definition as much as possible and, later, the rendering step, we have chosen to restrict our main framework to two dimensions. The optical properties of our fog, similarly to most other participating media rendering techniques, are proportional to its density, which depends itself on its extinction function. Therefore, the fog's main shape will actually be modeled as horizontal layers containing horizontal extinction function projected in a two-dimensional function basis. Further parameters, starting with a vertical extinction coefficient, will then thicken the fog vertically and give its final appearance.

Designing the fog's shape. Horizontal variations of our fog's density are modeled by specifying the value of each coefficient in the extinction function basis. These coefficients can be adjusted by hand, or be, for example, the result of a simulation, which was exported as a fogmap (see figure 3), i.e. a greyscale image, and then loaded back in our implementation.

Compared with other techniques such as RBF or particle-based methods, shaping our fog using a grayscale image is straightforward. The fogmap represents, in some extent, a direct preview of its aspect, what can be interesting for some applications where great intuition is needed. To ease the manual setting of the coefficients, we also developed a small application where the values of the density can be directly adjusted using a drag-and-drop interface.

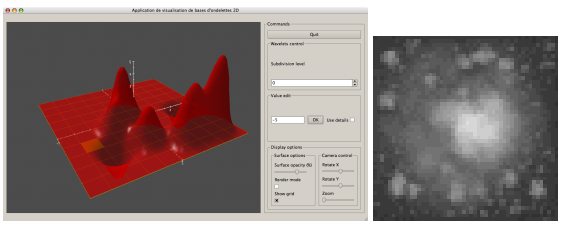

Figure 3: Left : snapshot of our modeling tool. Right : greyscale image representing the highest resolution coefficients

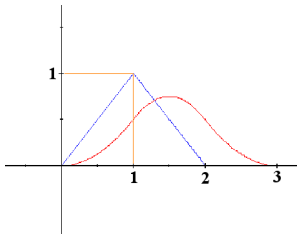

Figure 4: Shape of Haar, Linear and Quadratic B-Splines.

Choosing the basis functions. The appearance of the fog's density is a key criteria to choose our basis function. It is clear that abrupt changes in density would not look natural, so we would ideally like continuous functions to design smooth fogs using as few coefficients as possible. In order to avoid border effects, the scaling function must tend to zero on both sides of its support, which eliminates, for example, Legendre scaling functions.

For design and optimisation purposes, our rendering algorithm also needs the scaling function never to oscillate under zero. Whatever the trajectory of the ray within the function in 2D, and more generally within the fog, we would like to be sure that the sum of the density it intersects can only increase as it traverses the fog from the observer to the nearest object. Daubechies wavelets, which, by the way, are not symmetrical, might not be the way to go.

Finally, we have to consider the fact that, as will be discussed in the next section, the cost of using a particular type of wavelet is quadratically proportional to the support of the scaling function in one dimension.

According to their shape, the most adapted candidates seem the linear or quadratic B-Splines, which are shaped like a hill (see figure 4), and have a relatively compact support.

Although we are limited to wavelet scaling functions for the fog's representation, our method is not reduced to a particular type of wavelet. Our implementation specifically handles all degrees of B-Spline wavelets, but can be extended to other families, as long as they are compatible with Mallat's decomposition.

4.2 Preparing data for rendering

Generating multiple resolutions. One of our main goals is to take profit of multiresolution. Indeed, multiresolution helps to omit details that could be expensive to render, while being of limited visual importance. Therefore, perform a wavelet decomposition on our fog, which generate multiple level of details (i.e. multiple resolutions) from the original extinction function, and use them at the rendering phase. The most

adapted solution seems Mallat's fast wavelet transform, which is lossless, but requires data to be modeled in a scaling functions basis of the same type as the wavelets used for the decomposition. Therefore, each pixel of the fogmap will represent the coefficient of a scaling wavelet function.

Computing textures. From the fogmap we generate four multiple-level function bases : the approximation on a single level (i.e. a single 2D grid of values), and three different kinds of details for each level which was decomposed. All details bases have the same depth, which corresponds to the number of decomposition steps that were executed, value which must be decided by the user, depending on how much details can be omitted. Coefficients from the approximation and details basis will be stored in packed textures, and transmitted to the GPU under this form.

4.3 Rendering the fog

Overview. The purpose of our algorithm is to alter the original color of each pixel of the image using equation (3), blending L(P), the color of the object behind the fog, and the fog color to obtain L(O) the new color to compute.

For each pixel, we perform a ray-marching from the camera to the nearest surface, in which we integrate over the fog's extinction function to obtain the transmittance $\tau(O,P)$ along the view ray $\vec{OP}$.

The grid. As a result from the wavelet decomposition, the fog's density is scattered in several multiple-level function bases, having their own vector space and definition domain in 2D. Each single level can be assimilated to a rectangular grid, each cell being associated to both a coefficient and a basis function. Since all bases have the same definition domain, grids from different bases match at a given level.

Since our fog is only modeled in two dimensions, we do not take into account vertical variations and consider the fog as homogeneous on that direction. However, a vertical extinction coefficient taken as parameter allows to fade the fog out while its vertical distance from the viewer increases. But note that this is only a quick approximation over the exact equations.

Integration along the ray. The algorithm is iterative, but instead of advancing regularly along the ray, we move cell by cell. Each step corresponds to a new intersection between the ray and the grid, thus we always integrate between two intersections, i.e. between two positions on the perimeter of a square cell. This is a brute-force method, and some optimisations will be discussed in the next section.

We start by transposing both positions of the camera and the object from the scene to the fog's vector space. Our algorithm performs the entire integration level by level, and then, for each single function basis level, cell by cell.

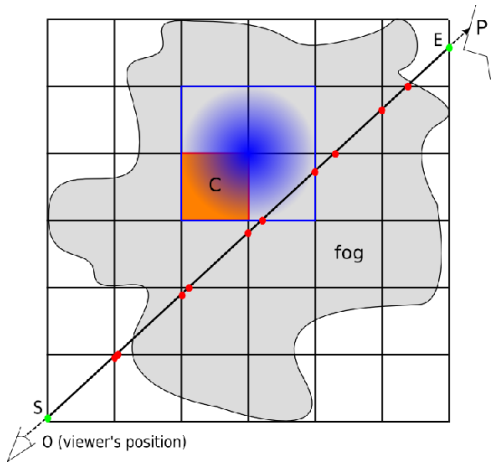

Figure 5: Ray-marching through a single level, designed with linear B-Splines scaling functions (support=2).

To initiate the integration on a given level, we first determine both entry and exit points of our integration on the grid. The entry point corresponds to either the nearest intersection between the ray and the current level's bounding box, or the viewer's position in case he stands within the fog. Similarly, the exit point corresponds to the intersection with either the farthest plane of the bounding box, or with the nearest object if situated within the fog.

When integrating a given level, the contribution of each single cell can be obtained by the product of both the function basis coefficient and the integral of the basis function associated to that cell along the view ray.

Mathematically, considering each cell c intersected by OP and using the extinction function decomposition of (10), we have :

$$\tau(O,P) = \sum_{cell:c} \int_{c\cap OP} K_t = \sum_{c} \left[\int_{c\cap OP} \alpha_c \phi\phi_c + \sum_{n=0}^{J-1} \int_{c\cap OP} \beta_c^n \phi\psi_c^n + \delta_c^n \psi\phi_c^n + \gamma_c^n \psi\psi_c^n \right] \quad (11)$$

J being the maximum decomposition level of our fog. Thanks to multiresolution analysis, each function indexed by cell c and level n is indeed a translated and scaled version of $\phi\phi$, $\phi\psi$, $\psi\phi$ or $\psi\psi$.

Therefore, we can precompute on the CPU a bunch of integrals for a set of sampled paths (complete or partial) within 1×1 squares on each function's definition domain, so that these values are directly available at runtime, transmitted on the GPU in packed textures. Integration on partial paths allow handling particular cases when the ray either starts and/or ends at the center of a cell within the fog's bounding box.

Figure 5 shows ray $\vec{OP}$ traversing a single level's grid from entry point S to exit point E. Integration steps (i.e. intersections with the grid) are shown in red. The

basis function (in this example : linear B-Spline scaling function) associated to the orange cell's coefficient c is shown in blue.

When using basis functions which are supported on an 1×1 square (e.g. Haar scaling functions and wavelets), their contribution area matches exactly that of the cell it is attached to, therefore we know that the cells which contribute to the pixel being rendered are exactly those traversed by the ray.

When the functions are supported on a domain larger than 1×1, part of the contribution of each cell gets superimposed on that of its neighbouring cells, thus also contributing to rays which do not necessarily pass through those cells themselves. Actually, a ray passing through a cell must take into account the contribution of that cell, plus the contributions of the $dx - 1$ previous cells on the X axis, times the $dy - 1$ previous cells on the Y axis, where dx and dy are the dimensions of the basis function's definition domain.

When the ray encounters a new function, we only integrate the density on the portion of that function which overlays the current cell, and then resume the integration for another 1×1 square of the same function when the ray traverses the next cell. If we directly integrate on the whole function's support at once, we omit the contributions of the functions attached to cells which are not encountered by the ray.

When the ending point has been reached, the whole process must be repeated with each level of each wavelet basis that was generated by Mallat's wavelet transform.

4.4 Optimizations & multiresolution

Our idea consists in omitting an increasing quantity of details from layers whom resolution is above a threshold which decreases as the observer moves away from the fog. When integrating the fog's density from the observer O to point P, the maximum integration distance $d_{\max_l}$ on level $l \in \mathbb{N}$ is given by :

$$d_{\max_l}(\vec{OP}) = \|OP\| \times \mu^l \tag{12}$$

where $\mu \in [0,1]$ is the optimization coefficient. When $\mu = 1$, the integration is performed entirely on all levels ; on the contrary, when $\mu = 0$, only the upper level of the basis is rendered.

As seen previously, when using scaling functions that are defined on more than an 1×1 square, the integration cost is no longer proportional to the fog's size, since more than each single particular cell traversed by the ray brings a contribution on these cell's area. That's why although the total number of coefficients modeling the fog stays almost unchanged, the rendering cost increases dramatically after the wavelet decomposition, since B-Spline wavelets always have a larger support than their scaling function.

Algorithm 1 Pseudo code of the shader

```
for each pixel do
  sum = 0
  for l = 0 to nb_levels do
    compute 2D entry point on grid
    compute 2D exit point on grid
    while pos ≠ exit do
      inter = compute next intersection with grid
      if (l = 0) then
        coef = get cell coef on approx basis
        approx = integrate on φφ between pos &
        inter
        sum += coef*approx
      end if
      coef = get cell coef on details1 basis
      det1 = integrate on φψ between pos & inter
      sum += coef*det1
      coef = get cell coef on details2 basis
      det2 = integrate on ψφ between pos & inter
      sum += coef*det2
      coef = get cell coef on details3 basis
      det3 = integrate on ψψ between pos & inter
      sum += coef*det3
      pos = inter
    end while
  end for
  pixel color=sum*obj color + (1-sum)*fog color
end for
```

When using such basis functions, for example linear or quadratic B-Splines, it can be interesting to use the two-scale relation for wavelets 8 to *deconstruct* the three wavelet bases. This turns them back into scaling function bases, which can then be merged (i.e. added) together. When using scaling functions with a large support, this operation, performed on the CPU just after the decomposition, can reduce the rendering cost by up to 2, while keeping the multi-resolution aspect brought by the decomposition. Moreover, if we perform a deconstruction, we can stop the integration as soon as the sum reaches a particular threshold, close to a great opacity. Deconstruction is important since it assures than each new cell will only add opacity.

5 RESULTS AND DISCUSSION

This algorithm has been implemented using GLSL, an Intel Core 2 Quad 2.8Ghz processor and a NVidia GeForce GTX 280 graphics card. Screen resolution is 800x600.

5.1 Performance

Table 1 show FPS results obtained when using our ray-marching alone to directly render raw Haar, linear or quadratic fogmaps, without any decomposition. Note also that a classical numerical integration along the ray,

Fog resolution	Haar	Linear	Quadratic
16×16	199	142	71
32×32	124	83	31
64×64	90	45	15

Table 1: FPS results with our ray-marching without optimizations.

Nb levels \ μ	1	0.8	0.6	0.4	0.2	0
0	45	-	-	-	-	-
1	35	39	47	55	66	83
2	31	45	55	71	90	124
3	27	39	66	76	99	166

Table 2: FPS results with our optimization, using a 64×64 linear B-Spline fogmap.

with 256 samples, runs at 99 FPS and suffers from severe aliasing artifacts. This is clearly outperformed by our method : Haar without decomposition gives 124 FPS and we can achieve similar FPS using a linear base. Each type of basis functions is defined on an area which size is increasing linearly in 1D, which involves a quadratically increasing number of neighbouring cells contributing to the density on each 1×1 square on the grid.

Table 2 show FPS results obtained when rendering a 64×64 linear B-Spline fog using our details dropping optimization, for different values of the tolerance parameter μ. With $\mu = 1$, no details are dropped, and we are performing a simple ray-marching. If, in addition, we do not apply any decomposition step, we are directly rendering the fogmap, like in table 1, therefore this value stands for the threshold above which we have a substential acceleration.

In table 3 we show, with arrows, the performance gain induced by our optimisations.

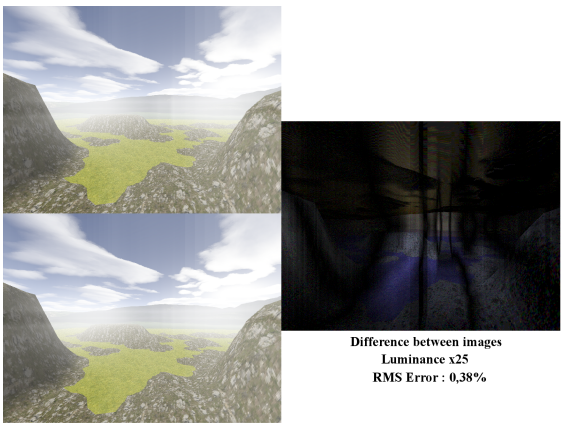

Figure 6: Quality difference when removing all layers of details (bottom left) from an Haar 32x32 (top left). Difference image (right) is shown (x25).

Nb levels	1	2	3
Linear 16x16	$58 \rightarrow 111$	$47 \rightarrow 99$	$35 \rightarrow 83$
Linear 32x32	$20 \rightarrow 62$	$15 \rightarrow 55$	$13 \rightarrow 49$
Quad. 32x32	$7 \rightarrow 23$	$5 \rightarrow 20$	$5 \rightarrow 18$

Table 3: FPS improvement when turning back into scaling function bases the four b-spline/wavelet generated by the decomposition (before $\rightarrow$ after).

5.2 Visual quality

The higher the degree of the B-Spline wavelet is, the smoother each basis function looks. With Haar wavelets, we can see, in figure 7.B, that the visual result is a bit unsatisfactory, with abrupt changes in density which betray the discontinuity of Haar functions. With linear B-Spline wavelets (figure 7.C) the framerate decreases but the visual result is a lot smoother and artifacts and peaks are now practically imperceptible. Finally, with quadratic B-Spline wavelets (figure 7.D), we loose in performance but this time, the quality gain is relatively low compared to linear B-Spline wavelets.

5.3 Discussion

Linear B-Spline seems a good trade-off between speed and quality but Haar could be used if rendering time is an issue. The advantage of using wavelets, beside their property of good data compression, is to have a mathematical representation of heterogeneous fog from physical simulation to rendering. Indeed, animating such fogs is easy, since wavelet decomposition can be performed in real-time. Moreover, unless previous approaches, we perform a precise numerical integration of density along the view ray, without any approximation. In comparison to particle approaches like [19], our method is more adapted to large outdoor scenes when camera is moving **in** the fog, and the modelling is far more intuitive than using particles.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented a new method for modeling heterogeneous fog using wavelet scaling functions. Rendering is performed through a simple decomposition scheme of the fog density function represented in a scaling function basis leading to sparse data. Wavelets and scaling functions allow and ease a certain number of precomputations, such as the integrals of the wavelets along each ray. A brute force rendering algorithm using the GPU has been presented allowing real-time rendering for moderated complex fog along with an optimized version taking profit of the sparcity of data induced by the wavelet decomposition. We have shown that our method outperforms brute force integration and allows exact computation of the effects of fog, without exotic approximations. Moreover, our method

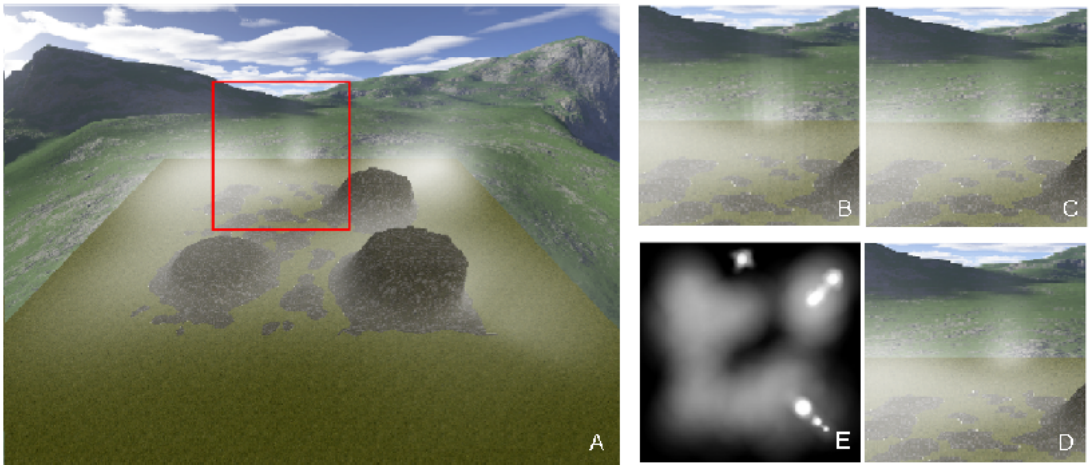

Figure 7: Quality difference with a large 30x30 fog. Fog taken from above (A), and the associated fogmap (E). Zoom on the red part when using Haar (B), Linear (C) and Quadratic (D) wavelets.

do not depends on the position of either the light or the fog, allowing simple transformations of the fog.

The use of wavelets opens the door to other major optimisations for our method. Mainly, the rendering algorithm can be improved by focusing only on the grid's cells which actually contain a non-negligible value, in order to be able to directly jump to the interesting zones of the fog when performing the integration along the ray. For this purpose, we aim at designing a simple GPU traversal of the graph generated by the wavelet decomposition. Since wavelets can be used to solve fluids equations, we also plan to link our rendering algorithm to a physical simulation involving wavelets, allowing a real-time physical animation and rendering of heterogeneous fog. Finally, we plan to add single scattering and volumetric shadows in our model.

REFERENCES

[1] V. Biri. Real Time Single Scattering Effects. In *Best Paper of 9th International Conference on Computer Games (CGAMES'06)*, pages 175 – 182, November 2006.

[2] Eva Cerezo, Frederic Perez-Cazorla, Xavier Pueyo, Francisco Seron, and François Sillion. A survey on participating media rendering techniques. *the Visual Computer*, 2005.

[3] R. Fedkiw, J. Stam, and H.W. Jensen. Visual Simulation of Smoke. In *proceedings of SIGGRAPH'01, Computer Graphics*, pages 15–22, August 2001.

[4] N. Foster and D. Metaxas. Modeling the motion of a Hot, Turbulent Gas. In *proceedings of SIGGRAPH'97, Computer Graphics*, pages 181–188, August 1997.

[5] Wolfgang Heidrich, Rüdiger Westermann, Hans-Peter Seidel, and Thomas Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 127–134, New York, NY, USA, 1999. ACM.

[6] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. Irradiance Gradients in the Presence of Participating Media and Occlusions. *Computer Graphics Forum (Proceedings of EGSR 2008)*, 27(4):xx–xx, 2008.

[7] H. W. Jensen and P.H. Christensen. Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps. In *Proceedings of SIGGRAPH'98, Computer Graphics*, pages 311–320, August 1998.

[8] J. Legakis. Fast multi-layer fog. In *Siggraph'98 Conference Abstracts and Applications*, volume Technical sketch, page 266, 1998.

[9] N.Adabala and S. Manohar. Modeling and rendering of gaseous phenomena using particle maps. *The Journal of Visualization and Computer Animation*, 11(5):279–293, December 2000.

[10] Nvidia. Fog polygon volumes - rendering objects as thick volumes, 2004.

[11] Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis light transport for participating media. In B. Peroche and H. Rushmeier, editors, *Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering)*, pages 11–22, New York, NY, 2000. Springer Wien.

[12] H. Rushmeier and K. Torrance. The zonal method for calculating light intensities in the presence of participating medium. In *proceedings of SIGGRAPH'87, Computer Graphics*, volume 21(4), pages 293–302, 1987.

[13] R. Siegel and J.R. Howell. *Thermal Radiation Heat Transfert*. Hemisphere Publishing, 3rd edition, 1992.

[14] F.X. Sillion. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. In *IEEE Trans. on Vision and Computer Graphics*, volume 1(3), pages 240–254, September 1995.

[15] J. Stam. Stable Fluids. In *proceedings of SIGGRAPH'99, Computer Graphics*, pages 121–128, 1999.

[16] E. J. Stollnitz, A. D. Derose, and D. H. Salesin. Wavelets for computer graphics: a primer.1. *Computer Graphics and Applications, IEEE*, 15(3):76–84, 1995.

[17] B. Sun, R. Ramamoorthi, S.G. Narasimhan, and S.K. Nayar. A practical analytic single scattering model for real time rendering. In *proceedings of SIGGRAPH'05, Computer Graphics*, volume 24 (3), pages 1040–1049, 2005.

[18] D. Zdrojewska. Real time rendering of heterogeneous fog based on the graphics hardware acceleration. In *proceedings of CESCG'04*, 2004.

[19] Kun Zhou, Qiming Hou, Minmin Gong, John Snyder, Baining Guo, and Heung-Yeung Shum. Fogshop: Real-time design and rendering of inhomogeneous, single-scattering media. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 116–125. IEEE Computer Society, 2007.