Real-Time Single Scattering with Volumetric Shadows

Biri Venceslas, Michelin Sylvain, Didier Arquès
University of Marne-La-Vallée
Institut Gaspard Monge
5, Boulevard Descartes, F-77454 Champs sur Marne, France
{arques,biri,michelin}@univ-mlv.fr

Abstract

The rendering of participating media is particulary difficult to handle because it depends on both camera position and lights positions. In an other hand participating media enhance greatly the realism of computer generated images. We present here a new algorithm that is able to render in real-time the single scattering induced by any participating medium. We use a new expression of the radiative transfer equation to compute efficiently the contribution of the participating medium in the illumination. We are able to manage not only the surface shadows but also the volumetric shadows of the scene. To achieve this, we render the shadow planes created by the light source and the silhouettes of objects in a correct order. It allows a quick rendering of participating media and volumetric shadows whitout aliasing. So this method can be used to compute quickly animations that represent scenes covered by a homogeneous participating medium.

1 Introduction

The rendering of participating media have always been a real challenge and especially when dealing with an anisotropic medium covering the whole scene. But its representation enhances greatly the realism of virtual scenes and it is indispensable for many simulations [21], including safety analysis for smoke, military and industrial simulations, entertainment, digital effects, driving and flying simulators...

Relying on physical equations [24], we can solve the illumination problem induced by such media. Both determinist and stochastic attempts are used in this way. Determinist methods encompass extensions of the radiosity algorithm, like the zonal method [22] and its improvements [25, 26]. Algorithms using spherical harmonics or discrete ordinates [1, 10, 15] belongs also to this group. Finally, other determinist methods use implicit representations of directional distribution of light [17, 27]. Stochastic methods include algorithms using Monte Carlo techniques [3, 9, 18] and more recent methods exploiting photon maps [7]. A detailed overview of most previous methods can be found in the Pérez et al. paper [19]. All these techniques produce very realistic images of participating media but, as they consider all the possible light interactions, they suffer of a long computation time.

Older methods [2, 8] focus on simpler light interactions, considering only single scattering. With this approach, and based on ray tracing, the algorithms of N.L. Max [14] and of T. Nishita and al. [16] belong to the fastest algorithms representing a scene with a participating medium covering a whole scene. But, when dealing with gas, fire or smoke, which are concentrated in some areas of the scene, it is necessary either to split these medium in many small participating media, or to cover them by a voxel grid. Methods [4, 5, 27] using voxel grids could simulate precisely the light exchanges in an area but are very slow. Blobs, each representing a small participating medium, can be used to speed up computation time. Several algorithms [20, 27, 23] use them to allow a fast rendering of located participating media like fire and smoke.

In this paper, we focus on single scattering and propose a new real-time rendering algorithm allowing the representation of a scene covered entirely by an anisotropic participating medium. Based on a new expression of the radiative transfer equation, this algorithm can represent efficiently and precisely not only the single scattering but also the surface shadows and the volumetric shadows without any aliasing.

In the next section, we review the radiative transfer equation governing illumination of participating media. In section 3, we focus on a new expression of this equation. Section 4 describes our approach and the corresponding algorithm. Finally section 5 presents our results and is finally followed by a conclusion in section 6.

2 Theoretical background

When an emission of light enters in a participating medium, the light is scattered through it. Different amounts of light depending on its density are then captured, reflected and emitted in all directions. In computer graphics, four interactions, illustrated in figure 1, are considered : absorption, emission, scattering and in-scattering.

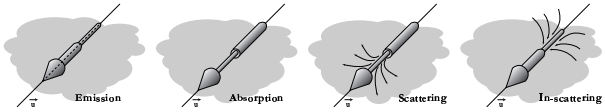

Figure 1. The four interactions between a light ray and a participating medium

As light travels along a ray of direction $\vec{u}$ in such medium, the radiance L is modified. 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 what happens in neon lighting for example. Scattering and in-scattering are the result of deviations in light direction induced by the medium. Scattering occurs when the incoming light is deflected from $\vec{u}$ in other directions, and in-scattering when light, after being deflected in the medium, follows $\vec{u}$. 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 $\vec{u}$.

The radiative transfer equation [24] – we will denote RTE – includes all these interactions in an unique equation expressing the radiance variation in a point *x* and in a direction $\vec{u}$:

$$\frac{dL(x,\vec{u})}{dx} = -K_t(x)L(x,\vec{u}) + K_t(x)J(x,\vec{u}) \qquad (1)$$

The first term represents the loss of light due to scattering and absorption. K_t is the extinction coefficient, sum of the absorption coefficient K_a and the diffusion coefficient K_s. The second term is called the source radiance, and represents the incoming energy in the direction $\vec{u}$. It is defined by :

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

where L_a is the eventual emission of light and p is the phase function expressing the ratio of light coming from any direction $\vec{\omega}$ which follows direction $\vec{u}$.

We will restrict our application to non emissive participating medium and also to single scattering. In this case, and using the notations from figure 2, the source radiance reduces in :

$$J(x,\vec{u}) = \frac{\Omega(x)}{4\pi}\frac{I_S(\vec{\omega})}{r^2}p(\vec{\omega},\vec{u})e^{-K_t r} \qquad (2)$$

where Ω is the albedo, i.e. the fraction between the diffuse coefficient and the extinction coefficient, and I the directional intensity of a point light source. For brevity, in this equation and in the following section, we will consider only one point light. But, equations can be easily generalized for more than one light.

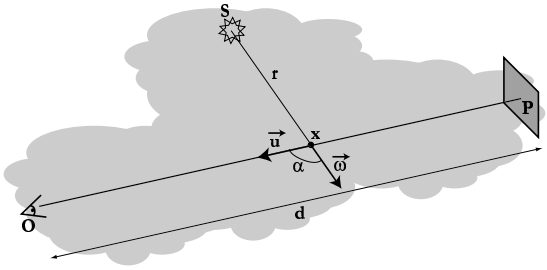

Figure 2. Notations used for single scattering

3 An angular formulation of the RTE

In this section we present an angular formulation of the RTE. This formulation, defined in [11], expresses explicitly the angle between the ray and the direction of light emission. Such a formulation can be used to ease further computations.

3.1 The new formulation

Instead of integrating the RTE regarding to the distance x along the ray, we choose to use the variation of the angle θ between the ray and the light direction (cf. figure 3). In the following, we will also consider that the participating medium covers the whole scene and that it is homogeneous. The integration of the RTE (1) along the ray gives the following expression, called the integral transfer equation :

$$\begin{aligned} L(O) &= L(P)e^{-K_t d} + \\ &\quad \frac{K_t\Omega}{4\pi}\int_0^d \frac{I_S(\vec{\omega})e^{-K_t x}e^{-K_t r}}{r^2}p(\vec{\omega},\vec{u})dx \end{aligned}$$

We know that $r = \sqrt{h^2 + (x-t)^2}$ and that $x = t + h.\tan(\theta)$. Using this variable change, we can obtain (see the annex) :

$$L(O) = L(P)e^{-K_t d} + \frac{K_t e^{-K_t t}\Omega}{4\pi h}L_m(P) \qquad (3)$$

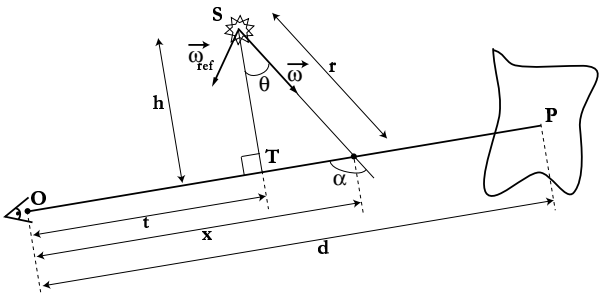

Figure 3. Notations for our new formulation

where :

$$L_m(P) = \int_{\theta_0}^{\theta_d} I_S(\theta+\beta) e^{-K_t \frac{\sin(\theta)+1}{\cos(\theta)}} p(\theta+\frac{\pi}{2}) dx$$

The first term of equation (3) will be called the surface contribution and the second one, the medium contribution.

This equation takes into account the possible intensity variation of the source depending on the emission direction. Our implementation use only homogeneous point light sources but spot light, directional light and real light sources can also be used. In the remaining section, we will continue with a general light source, but in our implementation we use only point light sources.

3.2 Approximation of the RTE

The kernel of the previous integral is complicated enough to prevent any analytic integration. But we can approximate this kernel to obtain a much more simple expression. The function in the kernel, we will denote Λ, depends only on the angle θ, considering that the extinction coefficient is constant. Then, we can develop its expression in a polynomial base :

$$\begin{aligned} \Lambda(\theta) &= e^{-K_t \frac{\sin(\theta)+1}{\cos(\theta)}} p(\theta+\frac{\pi}{2}) \\ &= c_0 + c_1\theta + c_2\theta^2 + o(\theta^2) \end{aligned} \quad (4)$$

For traditional phase functions – isotropic, hazy, murky... – the formal expressions of the coefficients c can be obtained (see the annex). We limit here the development to the second order but in practice we use superior orders.

We introduce equation (4) in the expression of $L_m(P)$ to obtain :

$$\begin{aligned} L_m(P) = \Bigg[& c_0 \int_{\theta_0}^{\theta^d} I(\theta+\beta)d\theta + \\ & c_1 \int_{\theta_0}^{\theta^d} \theta I(\theta+\beta)d\theta + ... \Bigg] \end{aligned} \quad (5)$$

Of course, for homogeneous point lights, theses integrals are easily computed :

$$L_m(P) = \left[c_0 \left[\theta\right]_{\theta_0}^{\theta^d} + c_1 \left[\theta^2/2\right]_{\theta_0}^{\theta^d} + ... \right] \quad (6)$$

and so we can obtain the contribution of the single scattering of point light source in constant time.

For more general light sources, whose intensity changes with the emission direction, it is often necessary to precompute these integrals. It can be done on N half circles sampled on the emission hemisphere (cf. figure 4). On every half circle, we sample also a set of M angles γ. Then, we can compute :

$$\Delta_i(\gamma) = \int_{-\frac{\pi}{2}}^{\gamma} \delta^i I(\delta) d\delta$$

where $\gamma = \theta + \beta$. Using this variable change, equation (5) can be written (see [11, 12] for a more detailed development) :

$$L_m(P) = \left[c_0' \int_{\delta_0}^{\delta^d} I(\delta)d\delta + c_1' \int_{\delta_0}^{\delta^d} \delta I(\delta)d\delta + ... \right]$$

And this equation can finally be expressed in function of the precomputed coefficients Δ :

$$\begin{aligned} L_m(P) = & [c_0' \left(\Delta_0(\gamma_0) - \Delta_0(\gamma_d)\right) + \\ & c_1' \left(\Delta_1(\gamma_0) - \Delta_1(\gamma_d)\right) + ...] \end{aligned} \quad (7)$$

A discussion on the quality of these approximations can be found in [12]. In general, they are quite good except when the ray passes close to the source, or when the observer is far away from the source. In the first case, the contribution is so high, and in the second case, so small, that these errors remain unnoticeable.

With this new formulation and these approximations, we are now able to compute in "constant time" – i.e. without any numerical integration – the contribution of single scattering along a lighted ray.

4 A new real-time algorithm

We present in this section an algorithm that is able to render a homogeneous but anisotropic participating medium that covers the whole scene. After a presentation of our approach, we will focus on the implementation allowing a fast rendering of the scene.

4.1 Our approach

According to the equation (3), for each pixel of the image, we must compute not only the luminance contribution

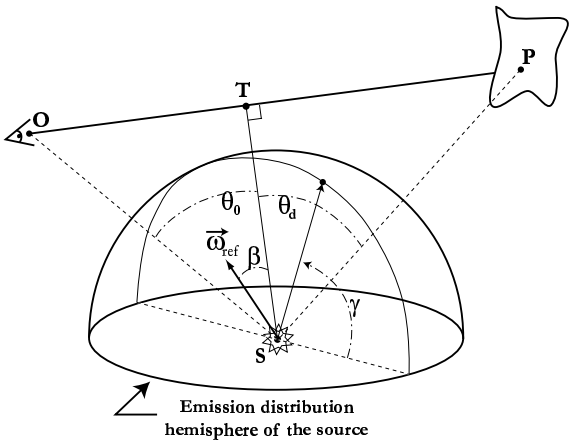

Figure 4. Half circles used for precomputations

of the point seen in the pixel, but also the contribution of the participating medium along the ray between the eye and this point. We can see already that our algorithm will have two main parts : first, the computation of the surface contributions, and second, the computation of the participating medium contributions.

The luminance in a point P of the scene, can be obtained using the equation :

$$L(P) = \rho_P \frac{I(\omega_P)\cos(\theta_P)}{\pi r_P^2} e^{-K_t r_P} \tag{8}$$

Here, two remarks have to be made. Firstly, with a homogeneous point light, the intensity $I(\vec{\omega})$ does not depend on the emission direction. Secondly, an ambient term can be used to represent multiple diffusion. In this case, the first term of equation (3) becomes :

$$L(P)e^{-K_t d} + I_a(1 - e^{-K_t d}) \tag{9}$$

Now, lets focus on the contribution of the participating medium along a ray. If the ray remains totally in the light, its contribution is straightforward to compute with our approximation of the RTE. But, as illustrated in the figure 5, the ray could be split into lighted and shadowed parts. In this case, the medium contribution along the ray is split into three parts on OA, BC and DP. The contribution of the single scattering of the ray OP is then :

$$L_m(P) = \left[\int_{\theta_D}^{\theta^P} \Lambda(\theta)d\theta + \int_{\theta_B}^{\theta^C} \Lambda(\theta)d\theta + \int_{\theta_0}^{\theta^A} \Lambda(\theta)d\theta \right] \tag{10}$$

The key idea of our approach is to rewrite this equation into a sum of differences. Indeed the light contribution of segment DP for example can be seen as the contribution of segment OP minus the one from segment OD. If we denote :

$$\Gamma(M) = \left[\int_{\theta_0}^{\theta^M} \Lambda(\theta)d\theta \right] \tag{11}$$

the equation (10) can now be written :

$$\begin{aligned} L_m(P) \quad = \quad & [(\Gamma(P) - \Gamma(D)) + \\ & (\Gamma(C) - \Gamma(B)) + \Gamma(A)] \end{aligned} \tag{12}$$

So if we are able to obtain the points A, B, C and D, we can compute the contribution of this ray using the previous equation and the ones seen in section 3.

Our basic approach is then : After having computed the surface contribution $L(P)$, we will compute all the unshadowed ray from the camera (O) to the lighted point in the scene (P). Then, using shadow planes obtained from the light position and the objects silhouettes, we will add or subtract luminance contribution in the ray depending on the orientation of the shadow plane. If the shadow plane is oriented toward the camera, its contribution will be added, and if not, it will be subtracted.

In the following, we will consider that the participating medium is homogeneous and recovers all the scene. Moreover, for simplicity, we will use only homogeneous point light source, even if more complicated source can be used. Finally our images will show only one light source even if several can be used.

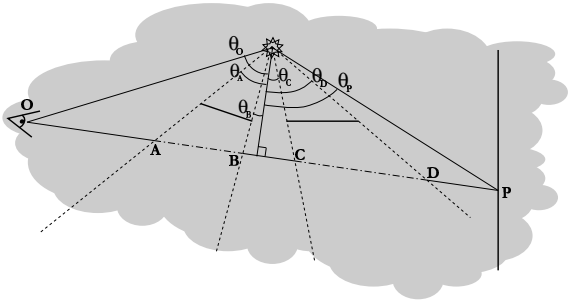

Figure 5. Case where a ray is partially in shadow

4.2 Silhouettes and shadow planes

In our algorithm, we need to draw the shadow planes of every objects in the scene. To obtain these planes, we will use the correct object silhouettes regarding to the light position. The silhouettes are computed in determining all the common edges of two neighbor triangles, one front-facing the light and the other back-facing it. Then all these edges are linked together, if possible, and stored in a list. The objective is to obtain a loop list, event if none loop list can be obtained and used - for non convex object for example.

A shadow plane is formed by a silhouette edge and the light position and is oriented toward the unshadowed area. Since we need the medium contribution on every point of the planes, we will mesh these planes, compute the contribution on every point of this mesh, and then, let OpenGL do the interpolation between these points. The mesh will be finer when close to the light, and coarser when far from it, for example, using a quadratic law. The need to compute the medium contribution of all shadow plane vertices involves to use the silhouette of global objects rather than the shadow planes of every little triangle constituting the objects. The figure 6 shows in dark green the silhouette of some objects of a scene, and in red and blue, respectively the front-facing and back-facing shadow planes of two objects.

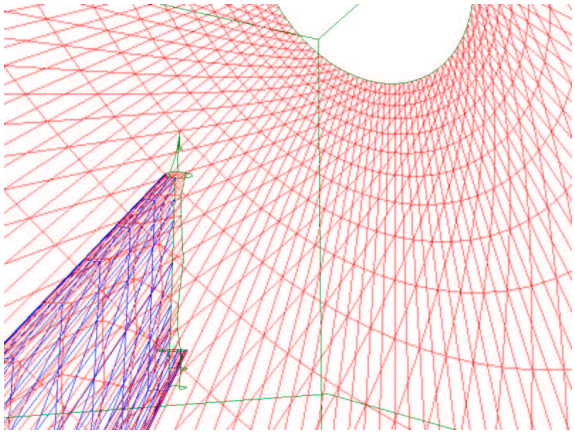

Figure 6. Silhouettes and shadow planes

4.3 Implementation

Our goal is to profit intensively of the graphic cards capacities and to use OpenGL to achieve real-time. The only computations we carry in software are the constructions of silhouettes and shadow planes, and, of course, the computations of the medium contributions. According to the section 4.1, our algorithm works in two main passes. The first pass is used to determine the surface shadows and contributions in the scene, when the second one computes the participating medium contributions and the volumetric shadows.

4.3.1 Surface contributions and shadows

A lot of efficient algorithms can be used to obtain the shadows in a scene. But the stencil algorithm [6] and its improvements fits perfectly with our approach. Yet instead of considering all the triangles of an object, we will use the silhouettes we have computed. This speed up this first pass and compensate for the determination of the silhouettes.

To simulate the absorption of the participating medium, we will first use the OpenGL fog function, allowing the graphic card to compute automatically the blending of equation (9). We also need to obtain, for a lighted point P in the scene its luminance contribution, defined in equation (8). To simulate it, we can use a vertex shader or, if it is not available, we can do a Taylor expansion of this expression around $r = 1$ (see the annex). To summarize, this first pass consists only in drawing the scene using the OpenGL lighting – and the stencil shadow algorithm – to obtain the proper surface contributions and shadows.

4.3.2 Participating medium contributions

The second pass is the most important one and splits up into two stages. The first stage determines the medium contributions on the surfaces, and the second one the shadow planes contributions. In the beginning of this second pass, we have the lighted zone of the image in the stencil buffer.

In the first stage, we must compute $\Gamma(P)$ for all lighted point P in the scene. For that, we need to disable the OpenGL lighting and the fog, and enable an additive blending. Then, we recover the camera position – used to compute $\Gamma(P)$ – and redraw all surfaces, maintaining the stencil test and giving to each vertex P the color computed with $\Gamma(P)$. Thanks to the stencil test, these contributions will only be written where the scene is lighted. In that stage the depth function have to be set to the equality.

In the second stage, we set the depth function to write only what lay in front of the surfaces composing the scene. Then, for each silhouette of each object, we draw its shadow planes, from the edges of the silhouette to the infinity. According to section 4.1, these shadow planes are constituted by fine mesh when they are close to the light, and coarse one when they are far from it. Depending on the orientation of the shadow plane, we use an additive blending for front-facing one, and negative blending for back-facing one. To avoid the drawing of shadow planes that are themselves in shadows, we need to render them in a correct order and use the stencil buffer. This ordering is the subject of the next section

4.4 Ordering and rendering of shadow planes

One other big problem is to avoid the drawing of shadow planes that are themselves in shadows. If we do not care about this problem, that will create an effect, that we call shadow in shadows, illustrated in figure 7. In the left image, we can see that the shadow of the top plane is propagated in the shadow of the bottom plane. Such effects are due to the fact that negative and positive shadow planes that are in shadow are also drawn. To prevent these artifacts, we should avoid to render the parts of the shadow planes that lay in shadows.

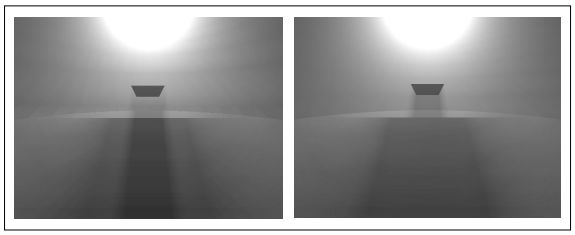

Figure 7. The shadow in shadows effect (left) and its correction (right)

The idea is to draw the shadow planes, back- or front-facing, in a front to back order and use the stencil buffer to avoid the drawing of front-facing planes when a front-facing planes have just been drawn and to do the same for the back-facing ones. So each time we construct a shadow plane, we also compute a distance and store the shadow plane in a list ordered with that distance.

The distance we use depends on the position of the shadow planes. We split the space with a plane which is orthogonal to the line camera-light (cf. figure 8). In the half space containing the camera position, the distance we use is :

$$l = d^2 \tag{13}$$

where l is the distance between the camera position and the infinite triangle composed by the point light and the edge of the silhouette. In the other half space, the distance is :

$$l = p^2 + \frac{1}{d^2} \tag{14}$$

where p is the distance between the light and the camera, and d is the distance between the camera and the image of the infinite triangle through the point light. So we are sure that planes from the opposite half space will be render after the first half space. With this distance, and attempting that the silhouettes are finely meshed, we will draw all the

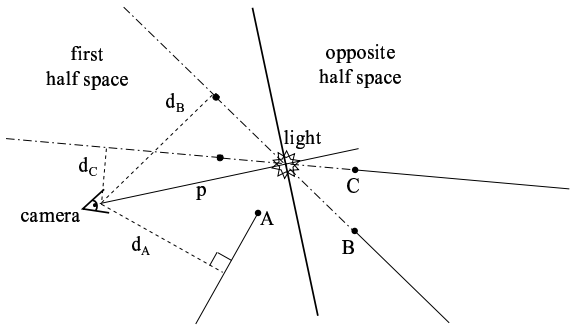

Figure 8. Distance used for shadow plane ordering

shadow planes in a front to back order. Then, when drawing a front-facing shadow plane, we will increment the stencil buffer, and when drawing a back-facing shadow plane, we will decrement the stencil buffer. The contribution of a front-facing shadow plane will be counted only if the stencil buffer have is initial value (0 for example). And the contribution of a back-facing one only when the stencil have is initial value plus one. In that case, we are sure of render only the part of shadow planes that are lighted.

4.5 Algorithm overview

In this section, we present our algorithm in the case of one light source. Dealing with more light sources is easy since all you need is to do the second pass for each light source. Of course, the stencil shadow algorithm must be modified in the same way. In the following code, and for brevity, we drop the gl and GL prefixes of OpenGL commands and tokens.

```
void display() {
  // Computation of changed silhouettes
  compute_sil();
  Clear(COLOR_BUFFER | DEPTH_BUFFER);
  MatrixMode(MODELVIEW);
  our_own_transformation();

  Enable(LIGHTING);
  if (USE_VERTEX_SHADER)
    lighting_using_vertex_shader();
  else
    compute_approximate_coefficient();
    standard_lighting();
  Enable(LIGHT0);
  Enable(FOG);
  Enable(CULL_FACE);
  Fogfv(FOG_COLOR,ambient_fog_color);
  Fogf(FOG_MODE,EXP);
  Fogf(FOG_DENSITY,coef_extinction);

  // First pass
  stencil_shadow_algorithm();

  // Second pass
  poscam = recover_camera_position();
  Disable(LIGHTING);
  Disable(FOG);

  // First stage : draw the surfaces
  DepthMask(FALSE);
  DepthFunc(EQUAL);
  Enable(BLEND);
  BlendFunc(GL_ONE,GL_ONE);
  Enable(STENCIL);
```

```
  draw_surf_cont(poscam,light);

  // Second stage : draw shadow planes
  Disable(CULL_FACE);
  DepthFunc(LESS);
  draw_shadow_plane(poscam,light);
}
```

The `compute_sil` function computes only the silhouettes that can have changed, i.e. the ones from moving objects. The function `draw_surf_cont` consists in computing, for all vertices of all objects, the participating medium contribution $L_m(P)$. To obtain them, we need to recover the angles θ_0, θ_d but also the distances h, d and t for each vertex. Then, the second part of equations (6) or (7) gives us the medium contribution in that particular vertex. The function `draw_shadow_plane` do the same for shadow planes to take into account the volumetric shadows. This function is the core of our approach detailed in section 4.1 and 4.4. It draws all the shadow planes using the ordered list described in section 4.4. When rendering, we use an additive blending function for front-facing shadow planes and a subtractive one for back-facing planes. We also use the stencil test as described in section 4.4.

5 Results and discussion

5.1 Results

To implement our method, we have designed a Qt application that uses OpenGL on a standard GeForce 2 graphic card and a 550 MHz processor. In this section we present some results obtained with this application. The first scene of figure 9 is a simple scene of a desk and two pencils – imported from a 3ds file – floating in the air above the desk. The figure 9.a shows the final image, including the participating medium contributions, and also the volumetric shadows under both pencils. Around the light we can see the single scattering of the isotropic medium. We also show in figure 9.b the same scene without the medium, to illustrate the realism enhancement achieved by integrating the single scattering.

The scene presented in figure 9 is a simple scene containing about 3000 triangles. It is rendered at more than 30 frames per second with the hardware described above. The figure 12 shows a snapshot from our animation. This scene contains more than 35 000 triangles – without the shadow planes – and is rendered at about 5 frames per second. Bigger color images and animations can also be found in [28]. Finally, we show on table 1 the ratio of work loads for each pass and stage. As expected, we can see that the computations of the shadow plane contributions represent the main cost of the whole process.

Scene of figure 12		30 triangles per shadow planes	10 triangles per shadow planes
Computation of silhouettes		1 %	1.7 %
1^{st} Pass		8.7 %	14 %
Ordering of shadow planes		9.6 %	16 %
2^{nd} Pass	surfaces	21.7 %	33.6 %
	volumes	59 %	34.7 %

Table 1. Work loads for each stage

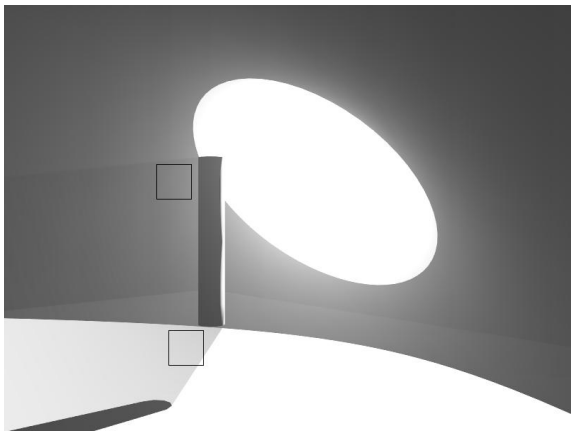

Figure 10. An example of the quantization problem

5.2 Quantization problem

In the figure 10, we point out a quantization problem. In that figure, we can see a extreme case where an object is very close to the point light. In the bottom square, the participating medium creates a pronounced shadow that is incorrect. As pointed out in the figure 11, this problem is due to the crop of OpenGL. When we add the first front-facing shadow plane, the luminance of the pixel is already close to one and OpenGL crop the real value to one. So

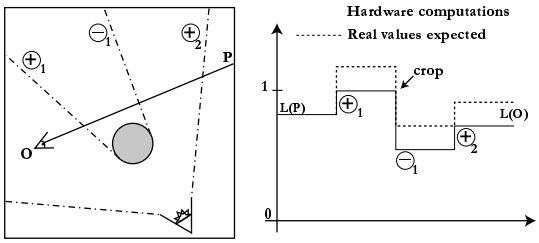

Figure 11. Illustration of the quantization problem

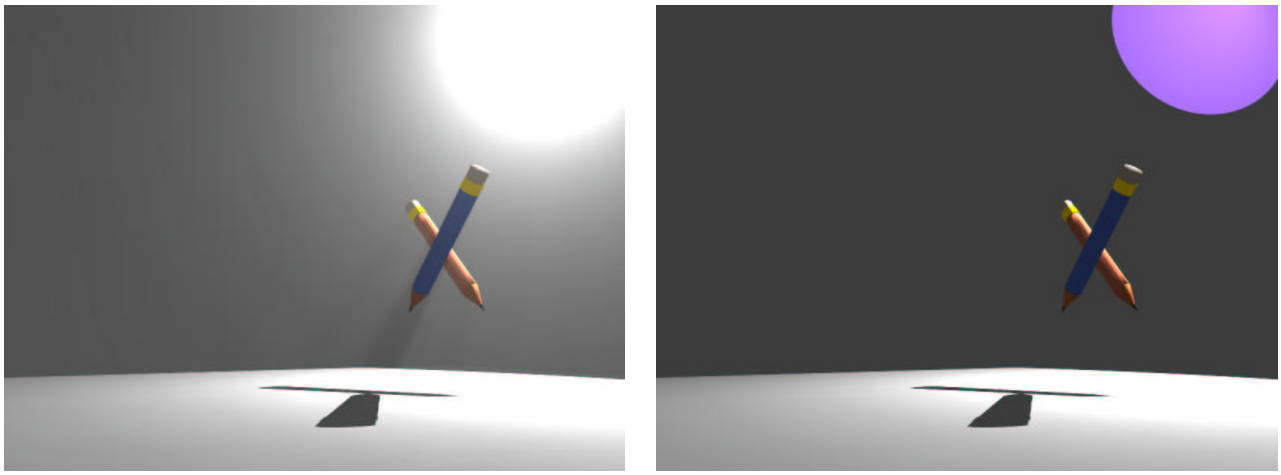

Figure 9. a. Image from a simple scene c. Same scene without participating medium

the raise due to the medium contribution is ignored. Then, when drawing the first back-facing shadow plane, the contribution decreases too mush compared to the real value (cf. figure 11). In the same time, the computation of the medium contribution remains correct in the top square.

One way to solve the problem is to make the second stage (of the second pass) before the first stage. In fact, it is a more logical order since we draw all medium contributions from the eye to the surfaces. But it involves to compute again the surface shadows with the stencil shadow algorithm. And unfortunately, this technique will not work all the time, and when a point will have a color close to 1, we will always have a quantization error. Solving the quantization problem involves the use of a auxiliary or extended buffer to store the correct light variations.

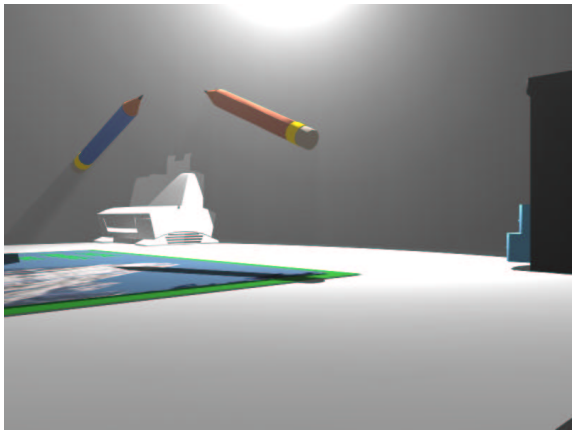

Figure 12. A snapshot from our animation

6 Conclusion and future works

We presented in this paper a new real-time algorithm that is able to compute the single scattering of a participating medium. Our algorithm is fast enough to handle more than 30 frames per second for simple scenes, which is a major improvement over other single scattering algorithms, especially for a medium covering the whole scene. But the main point of our algorithm is that it can take full profit of the capacities of the graphic cards. As outlined above, the only computations that we have done in software are the participating medium contributions and the ordering and the computation of shadow planes. But complex vertex shaders – or fragment shaders if possible – can be used to make the graphic card computes the medium contributions. Moreover, the quantization problem may be avoided using fragment shaders or an auxilliary buffer. We also have to point out that our algorithm does not create any aliasing effect, for surface shadows and volumetric shadows, thanks to the use of the exact shadow planes.

Apart from the resolution of the quantization problem, several developments can be done to improve the algorithm. Firstly, a clustering approach can avoid, for small objects constituted by many vertices, to compute the same contribution for a lot of neighboring vertices. Secondly, simple occlusion queries will also prevent to compute the medium contribution for surfaces and shadow planes that are out of focus or certainly in shadow. Both clustering and culling approachs will greatly speed up this already fast algorithm.

References

[1] N. Bhate and A. Tokuta. Photorealistic Volume Rendering of Media with Directional Scattering. 3^{rd} *Eu-*

rographics Workshop on Rendering, pp. 227–245, May 1992.

[2] J.F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics, proceeding of SIGGRAPH'82*, vol. **16**(3), pp. 21–30, July 1982.

[3] P. Blasi, B. LeSac and C. Schlick. An Importance Driven Monte-Carlo Solution to the Global Illumination Problem. 5^{th} *Eurographics Workshop on Rendering*, pp. 173–183, June 1994.

[4] N. Foster, D. Metaxas. Modeling the Motion of a Hot, Turbulent Gas *Computer Graphics, proceeding of SIGGRAPH'97*, pp. 181–188, August 1997.

[5] R. Fedwik, J. Stam, H. Wann Jensen Visual Simulation of Smoke *Computer Graphics, proceeding of SIGGRAPH'01*, pp. 15–22, August 2001.

[6] T. Heidman. Real Shadows Real Time, *IRIS Universe*, vol. 18, pp 28-31, 1991.

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

[8] J.T. Kajiya and B.P. von Herzen. Ray tracing volume densities. *Computer Graphics, proceeding og SIGGRAPH'84*, vol. **18**(4), pp. 165–174, July 1984.

[9] E.P. Lafortune and Y. Willems. Rendering Participating Media with Bidirectional Path Tracing. 6^{th} *Eurographics Workshop on Rendering*, pp. 92–101, June 1996.

[10] E. Languénou, K. Bouatouch and M. Chelle. Global Illumination in Presence of Participating Media with General Properties. 5^{th} *Eurographics Workshop on Rendering*, pp. 69–85, June 1994.

[11] P. Lecocq, S. Michelin, D. Arques and A. Kemeny. Mathematical approximation for real-time rendering of participating media considering the luminous intensity distribution of light sources. *Proceeding of Pacific Graphics 2000*, pp. 400–401, 2000.

[12] P. Lecocq. Simulation d'éclairage temps réel par des sources lumineuses mobiles et statiques : outils pour la simulation de conduite. *phD Thesis of the University of Marne la Vallée*, 2001.

[13] J. Legakis. Fast multi-layer fog. *Siggraph '98 Conference Abstracts and Applications*, pp. 266, Technical Sketch, 1998.

[14] L.N. Max. Atmospheric Illumination and Shadows. *Computer Graphics, proceeding of SIGGRAPH'86*, vol. **20**(4), pp. 117–124, august 1994.

[15] L.N. Max. Efficient Light Propagation for Multiple Anisotropic Volume Scattering. 5^{th} *Eurographics Workshop on Rendering*, pp. 87–104, June 1994.

[16] T. Nishita, Y. Miyawaki and E. Nakamae. A Shading Model for Atmospheric Scattering considering Luminous Distribution of Light Sources. *Computer Graphics, proceeding of SIGGRAPH'87*, vol. **21**(4), pp. 303–310, 1987.

[17] T. Nishita, Y. Dobashi and E. Nakamae. Display of Clouds Taking into Account Multiple Anisotropic Scattering and Skylight. *Computer Graphics, proceeding of SIGGRAPH'96*, pp. 379–386, June 1994.

[18] S.N. Pattanaik and S.P. Mudur. Computation of Global Illumination in a Participating Medium by Monte-Carlo Simulation. *The Journal of Vis. and Comp. Animation*, vol. **4**(3), pp. 133–152, 1993.

[19] F. Pérez, X. Pueyo and F.X. Sillion. Global Illumination Techniques for the Simulation of Participating Media. 8^{th} *Eurographics Workshop on Rendering*, June 1997.

[20] W.T. Reeves. Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *Computer Graphics, proceeding of SIGGRAPH'83*, vol. **17**(3), pp. 359–376, July 1983.

[21] H.E. Rushmeier. Rendering Participating Media : Problems and Solutions from Application Areas. 5^{th} *Eurographics Workshop on Rendering*, pp. 35–56, June 1994.

[22] H.E. Rushmeier. and E. Torrance. The Zonal Method For Calculating Light Intensities in the Presence of a Participating Medium. *Computer Graphics*, vol. **21**(4), pp. 293–302, July 1987.

[23] J. Stam, E. Fiume. Depicting Fire or Other Gaseous Phenomena Using Diffusion Process. *Computer Graphics, proceeding of SIGGRAPH'95*, pp. 129–136, August 1995

[24] R. Siegel and J.R. Howell. *Thermal Radiation Heat Transfert*. 3rd ed. Hemisphere Publishing, Washington, 1992.

[25] F.X. Sillion. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. *IEEE Trans. on Vis. and Comp. Graphics*. vol. **1**(3), pp. 240–254, Sept 1995.

[26] L.M. Sobierajski. *Global Illumination Models for Volume Rendering*. Chapter 5: Volumetric Radiosity, pp. 57–83, PhD Thesis, 1994.

[27] J. Stam. Multiple Scattering as a Diffusion Process. 6^{th} *Eurographics Workshop on Rendering*, pp. 41–50, June 1994.

[28] http://www-igm.univ-mlv.fr/~biri/index.html

Annexes

Obtention of the new formulation of the RTE

The variable change is :

$$\left| \begin{array}{l} x = t + h\ \tan(\theta) \\ du = h(1+\tan^2(\theta))d\theta \end{array} \right.$$

Lets write the integral term of equation (3) :

$$G = \int_{\theta_0=\tan^{-1}(\frac{-t}{h})}^{\theta_d=\tan^{-1}(\frac{d-t}{h})} \Lambda$$

with

$$\begin{aligned} \Lambda &= \frac{\exp\left[-K_t\left(t+h\tan(\theta)+\sqrt{h^2(1+\tan^2(\theta))}\right)\right]}{h^2(1+\tan^2(\theta))} \\ &\quad .\left[h(1+\tan^2(\theta))I_S(\theta+\beta)p(\theta+\frac{\pi}{2})d\theta\right] \\ \Lambda &= \frac{\exp\left[-K_t\left(t+h\tan(\theta)+\sqrt{h^2(1+\tan^2(\theta))}\right)\right]}{h} \\ &\quad .\left[I_S(\theta+\beta)p(\theta+\frac{\pi}{2})d\theta\right] \end{aligned}$$

and finally :

$$G = \frac{e^{-K_t t}}{h}\int_{\theta_0}^{\theta_d}\Lambda$$

with

$$\begin{aligned} \Lambda &= \exp\left[-K_t\left(h\tan(\theta)+\sqrt{h^2\frac{1}{\cos^2(\theta)}}\right)\right] \\ &\quad .\left[I_S(\theta+\beta)p(\theta+\frac{\pi}{2})d\theta\right] \end{aligned}$$

which can be simplified in :

$$G = \frac{e^{-K_t t}}{h}\int_{\theta_0}^{\theta_d} e^{-K_t h(\frac{sin\theta+1}{cos\theta})} I_S(\theta+\beta)p(\theta+\frac{\pi}{2})d\theta$$

Coefficient c for traditionnal phase functions

Isotropic phase function :

$$\begin{aligned} c_0 &= e^{-K_t h} \\ c_1 &= -K_t h e^{-K_t h} \\ c_2 &= \left(K_t^2h^2 - K_t h\right)\frac{e^{-K_t h}}{2} \\ c_3 &= \left(\frac{K_t^2h^2}{2} - \frac{K_t h}{3} - \frac{h^3K_t^3}{6}\right)e^{-K_t h} \\ c_4 &= \left(\frac{11K_t^2h^2}{24} - \frac{5K_t h}{24} - \frac{h^3K_t^3}{4} + \frac{h^4K_t^4}{24}\right)e^{-K_t h} \\ &\ldots \end{aligned}$$

Hazzy phase function :

$$\begin{aligned} c_0 &= \frac{265}{256}e^{-K_t h} \\ c_1 &= \left(-\frac{265K_t h}{256} + \frac{9}{36}\right)e^{-K_t h} \\ c_2 &= \left(\frac{265K_t^2h^2}{512} - \frac{121K_t h}{512} + \frac{63}{64}\right)e^{-K_t h} \\ &\ldots \end{aligned}$$

Murky phase function :

$$\begin{aligned} c_0 &= \frac{2147483673}{2147483648}e^{-K_t h} \\ c_1 &= \left(\frac{2147483673K_t h}{2147483648} - \frac{25}{67108864}\right)e^{-K_t h} \\ c_2 &= \left(\frac{2147483673K_t^2h^2}{4294967296}\right. \\ &\quad \left. -\frac{2147482073K_t h}{4294967296} + \frac{775}{134217728}\right)e^{-K_t h} \\ &\ldots \end{aligned}$$

Approximate coefficient for lighting

The classical lighting function of OpenGL as the form :

$$L(P) = \rho_P \frac{I\cos(\theta)}{a_0 + a_1 r + a_2 r^2} \tag{15}$$

According to an Taylor expansion of expression (8), we can approximate it with the coefficients :

$$\left| \begin{array}{l} a_0 = \left(K_t + \frac{K_t^2}{2}\right)e^{K_t} \\ a_1 = \left(-3K_t - K_t^2\right)e^{K_t} \\ a_2 = \left(1 + 2K_t + \frac{K_t^2}{2}\right)e^{K_t} \end{array} \right. \tag{16}$$