HAL
open science

Real Time Single Scattering Effects

Venceslas Biri

To cite this version:

Venceslas Biri. Real Time Single Scattering Effects. 9th International Conference on Computer Games (CGAMES'06), Nov 2006, France. pp.175-182. hal-00622215

HAL Id: hal-00622215

https://hal.science/hal-00622215

Submitted on 22 Mar 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Real Time Single Scattering Effects

Venceslas Biri
Gaspard Monge Insitute - University of Marne-la-Vallée
6 cours du Danube,
F-77700 SERRIS
Email : biri@univ-mlv.fr

***Abstract*— Rendering mist, haze or fog remains a challenge in current computer graphics since it is intrinsically a 3D problem. While the attenuation caused by fog is easy to implement, single scattering effects such as glows and shafts of light, that increase considerably the realism, are harder to render in real-time. This paper addresses the rendering of such effects along with volumetric shadows induced by shadow casters in the participating media. Whereas techniques related to shadow maps have been explored when rendering with single scattering, this paper proposes a real-time algorithm using the philosophy of shadow volumes, including volumetric shadows. With a spatial coherence method, simple shaders and an intensive use of the stencil buffer, we render the shadow planes in a back to front order to obtain the correct volumetric shadows. Therefore our method is easy to integrate in a graphics engine using the shadow volume technique since it requires only a little additional texture memory and is implemented with simple shaders. Realistic images can be produced in real-time for usual graphic scenes and at a high level framerate for complex scenes, allowing changes in the properties of participating medium, animations of objects and even light sources movements.**

***Keywords*— Single scattering, Real Time, Hardware rendering**

I. Introduction

If light scattering has been intensively explored, from Blinn in 1982 [1] to Sun et al. [2], it is mainly because it greatly enhances the realism of virtual scenes and can greatly improve the graphic quality of computer games. Indeed, the light scattering occurs everywhere in a scene and therefore is intrinsically a 3D phenomenon. For example, look at the differences between the Figure 1.a rendered classically and the Figure 1.b where the contributions of a participating medium have been added. In the first picture, it is not clear where the lights are while they can be roughly located in the second. Nevertheless, the Figure 1.c shows that, in some cases, rendering only the participating medium is not enough. In order to increase realism, we must render the volumetric shadows. Volumetric shadows are the shadows that cast objects in the participating medium itself. They greatly contribute on the presence in a virtual world. Here, the representation of shadow volumes is necessary to obtain a realistic image and understand clearly that a light stands inside the amphora.

A simple and common way to model light scattering is to handle the attenuation due to participating medium and consider the multiple scattering of light as homogeneous and constant over the scene. This is the OpenGL fog model which is popular since it only needs a fog color and is handled automatically by graphic cards. Figure 1.a is a perfect example of this technique. Despite its great speed, this model poorly represents all the effects induced by light scattering. On the other hand, methods seeking for great realism have investigated the computation of the multiple scattering of light through the medium. But due to this goal, these methods are very slow since they need Monte Carlo or finite element techniques.

In this paper we investigate the complete single scattering illumination model. In this local illumination approach, only the first scattering of light in the medium is taken into account. The remaining scattering of light is considered as homogeneous and constant all over the scene, like in the OpenGL fog model. The originality of the model is the introduction of the **indirect single scattering**. Indeed, we will call **direct single scattering** the effect of the first scattering of light along view rays. The **indirect single scattering** is the same effect but along illumination rays of any point of scene. Our contributions in this paper are :

- **Define an analytic and comprehensive formulation of light scattering along view rays and illumination rays**. Based on a angular formulation of the radiative transfer equation, we present a way to use precomputed 2D table to compute directly these contributions.
- **Integration of the volumetric shadows**. We build a method based on the shadow volume technique and using spatial coherence strategy, allowing the rendering of volumetric shadows in the scene, especially discernable around light sources.
- **Hardware implementation**. Except for the determination of object's silhouette, all the work is done in hardware. We store our precomputed 2D tables in textures and use simple shaders to render the illumination of objects and participating media. The rendering of volumetric shadows also involves an intensive use of the stencil buffer.

Our goal is to design an algorithm that can render accurately participating media, including effects like light beams in foggy or smoky scenes. Our method is not restricted to isotropic participating media which can be lit by one or several, static or moving, point light sources since no precomputation is done involving either lights or camera. Our technique produces high resolution images and takes into account volumetric shadows, cast by occluders contained in the media. With very few

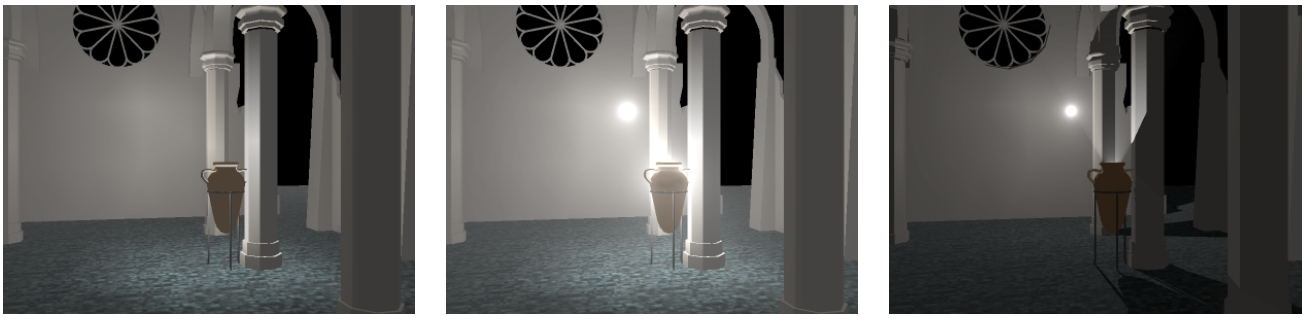

Fig. 1. The same scene lit a. (left) classically, b. (center) with single scattering and c. (right) with single scattering and volumetric shadows. Who could say precisely where are the lights in the left picture ? This is obvious in the right picture

texture memory cost, but using intensively graphics hardware, our method can render images at a high frame rate and is real-time for classical graphics scene. Our method is also easy to implement in traditional graphics engines since it follows the same strategy than the shadow volume algorithm, and use only shaders and textures. Therefore, it is straightforward with our method to obtain animations where objects or even light sources can move.

II. Related Work

The representation of participating media has been a real challenge for years. We can easily divide all these studies between the single and the multiple scattering methods. The last ones try to compute all light reflections and inter-reflections inside the medium, whatever the number of these reflections. Despite their realism, they suffer from excessive computation time due to the complexity of light exchanges occurring in these cases. Therefore these approachs are not suitable for our goal and we will focus on methods considering only the single scattering case.

These techniques [9], [12], [15]–[17] approximate multiple reflections of light as a constant ambient term and consider only the first scattering of light ray in the direction of the camera. This assumption allows a direct rendering of the illumination of the medium which is more suitable for interactive rendering. Visualization is often done by ray tracing or ray marching. View rays are followed to gather the participating media contributions. Unfortunately, these methods [18], [19] are far from being real-time on a conventional desktop computer. With the growing capacities of graphics hardware, the real-time problem has been investigated.

Two approaches can be used to achieve this goal : volume rendering or direct representation. In order to add the volumetric shadows, the first approach will naturally use shadow maps techniques whereas the second is implicitly shadow volumes oriented [20]. Volume rendering is a classic solution to render a participating medium which is a volume de facto. Methods like [3]–[7] represent density or illumination in voxels encoded into 2D or 3D textures. Accumulation techniques using textured slices or virtual planes are then used to display the result. These methods could produce nice images of clouds or gas, but, in addition to requiring a lot of texture memory, they are not suitable for shafts of light where sharp edges exist. Special methods are defined to render beams and shafts of light precisely and most of them [11]–[14] use volume rendering techniques along with sampling shadows in shadow maps. Unfortunately, they suffer from artifacts due to the sampling. Dobashi et al. [15] present a very elegant solution to solve this problem using specialized adaptive sampling for shadows. They obtain an interactive rendering of participating media without aliasing or artifacts. However the image resolution remains low since the method is expensive in terms of fillrate. Moreover, the method works only with static lights due to the precomputation of shadow maps and only addresses the direct single scattering.

Method	Volume rendering	Direct computation
Direct single scattering	[3] [4] [5] [6] [7]	[8] [9] [10]
+ Indirect single scattering	none	our method and [2]
+ Volumetric shadows	[11] [12] [13] [14] [15]	our method

TABLE I

Overview of previous work on single scattering

The algorithms belonging to the second approach compute directly on every point of the scene the contribution of the participating medium. It is well adapted to classical graphics engines since it consists in one more rendering of the scene. In this case, methods like [8], [9] use participating medium boundaries, or special virtual planes, combined with vertex and fragments shaders. Other methods focus on the rendering of the atmosphere [10]. Despite their speed and simplicity, all these methods consider only the direct single scattering. The most advanced method of this group is proposed by Sun et al. [2] and is the first to consider the effects of the indirect single scattering. Unfortunately, it does not take shadows into account, even though it is real-time. Our work belongs to this group but is the only one of them to integrate direct single scattering, indirect single scattering and volumetric shadows. Compared to our own work presented in [21], we present here an hardware implementation along with the representation of the indirect single scattering which are major improvements.

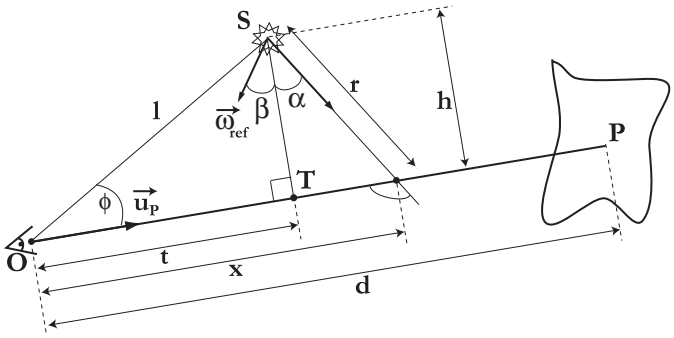

Fig. 2. Notations for our model

III. The single scattering illumination model

In this section, we will present an analytic formulation of the single scattering illumination model. We will mainly focus on in-scattering since absorption, emission, multiple scattering and out-scattering are really simple to integrate. We start in the following developments with the angular formulation we presented in [21], [22], and that were finely used by [2], work we review and extend here to obtain the complete illumination model of single scattering. Our developments consider only homogeneous participating media defined all over the scene and point light sources, assumed to be isotropic.

A. Single scattering along a ray

Considering a view ray immersed in a participating medium, the radiance L observed along $\vec{u_P}$ from a point P can be written (see Figure 2) :

$$L_{\vec{u_P}} = L_{dss}(\vec{u_P}) + e^{-k_t OP} L(P) + L_a e^{-k_t OP} + L_{oiss}(\vec{u_P}) \quad (1)$$

where L_{dss} is the direct single scattering and $L(P)$ the radiance of P attenuated by both absorption and out-scattering. k_t is the extinction coefficient, sum of the absorption coefficient and the diffusion coefficient, properties of the participating medium. The third term represents both emission and multiple scattering inducing a constant radiance L_a along the ray. The last term is the object indirect single scattering.

Now we focus on the radiance of P that can be written :

$$L(P) = L_{iss}(P) + e^{-k_t SP} L_d(P) + L_a e^{-k_t SP} \quad (2)$$

L_{iss} is the indirect single scattering received on point P and $L_d(P)$ an usual direct illumination, like Phong model, that is attenuated by absorption and out-scattering. The third term is once again the contribution of emission and multiple scattering on the illumination ray.

Then, only L_{dss} and L_{iss} remain unknown. They both are the contribution of the first light scattering along, respectively, a view ray and an illumination ray. First of all, we will investigate the general contribution of this first scattering along a ray before to see its application on view rays and illumination rays. The integral transfer equation, presented in [23], gives the incoming radiance L_{ss} in direction $\vec{u_P}$ stopped at vertex P and seen from O (see Figure 2) :

$$L_{ss}(\vec{u_P}) = \frac{k_t \Omega}{4\pi} \int_0^d \frac{I_S(\alpha+\beta) e^{-k_t x} e^{-k_t r}}{r^2} p(\alpha + \frac{\pi}{2}) dx \quad (3)$$

In this equation x, r, d, h are geometrical factors and Ω is the albedo, i.e. the fraction between the diffuse coefficient and the extinction coefficient of the medium. p is the phase function expressing for any incoming direction the ratio of light following the ray direction. Finally, I_s is the directional intensity of the source computed relatively to the reference direction $\vec{\omega_{ref}}$. Equation (3) expresses the in-scattering which is responsible for the subtle effects of atmospheric scattering.

The previous equation can be simplified [21], [22] using the angle α to minimize the dependency of the integral to it. Using the variable change $x = t + h.\tan(\alpha)$, t and h are constant along the ray, we can obtain (see the annexes) :

$$L_{ss}(\phi, d, l) = \frac{k_t \Omega e^{-k_t t}}{4\pi h} \int_{\gamma_0}^{\gamma_d} I_S(\alpha+\beta) p(\alpha+\frac{\pi}{2}) e^{-k_t h \frac{\sin(\alpha)+1}{\cos(\alpha)}} d\alpha \quad (4)$$

where

$$\begin{aligned} \gamma_0 &= -\frac{\pi}{2} + \phi = atan(\frac{-t}{h}) \\ \gamma_d &= atan(\frac{d - lcos(\phi)}{lsin(\phi)}) = atan(\frac{d-t}{h}) \end{aligned}$$

B. The direct single scattering

In the case of view rays, the point O is the position of the camera. Therefore, the distance between O and S remains constant whatever the view ray considered. The only variables in the equation (4) are the angle ϕ and the distance d. For an isotropic light, this equation becomes :

$$L_{dss}(\phi, d) = \frac{k_t \Omega I_S e^{-k_t t}}{4\pi h} \int_{\gamma_0}^{\gamma_d} p(\alpha+\frac{\pi}{2}) e^{-k_t h \frac{\sin(\alpha)+1}{\cos(\alpha)}} d\alpha \quad (5)$$

If we denote :

$$\Gamma(\lambda, \gamma) = \int_0^{\gamma} p(\alpha+\frac{\pi}{2}) e^{-k_t \lambda \frac{\sin(\alpha)+1}{\cos(\alpha)}} d\alpha \quad (6)$$

the direct single scattering could be written :

$$\begin{aligned} L_{dss}(d, \phi) &= \frac{k_t^2 \Omega I_S}{4\pi} \frac{e^{-k_t t}}{k_t h} \Big[\Gamma(k_t h, atan(\frac{d-t}{h})) \\ &\quad - \Gamma(k_t h, atan(\frac{-t}{h})) \Big] \quad (7) \end{aligned}$$

This particular formulation is well suited to optimize the shader that computes the direct single scattering. Note that the 2D function Γ is purely numerical and only depends on the shape of the phase function. It can therefore be precomputed and stored in a 2D table. Figure 3 shows two examples for isotropic phase function and mie-hazzy phase function.

C. The indirect single scattering

Inspired by [2], we also compute the illumination of vertex P due to the light scattering all over the scene. The indirect single scattering can be written (see Figure 4) as follows :

$$L_{iss}(P) = \int_{\theta=0}^{2\pi}\int_{\phi=0}^{\pi} L_{ss}(\phi, d_{\phi,\theta}, l) f_r(\vec{u_c}, \vec{u_v})(\vec{u_c}.\vec{n}) d\omega \quad (8)$$

The first difficulty is to obtain for each direction $\vec{u_c}$ the distance $d_{\phi,\theta}$. This distance has been arbitrary fixed at ∞ in [2] but this is a coarse approximation. Nevertheless, it is necessary if we want to avoid too much complexity. So we will make the same assumption and take, $d_{\phi,\theta} = \infty, \ \forall \phi, \theta$.

First of all, consider the Lambertian case where the function f_r is the diffuse reflectivity of the surface k_d. We then have :

$$L_{iss}(P) = k_d \int_{\theta=0}^{2\pi}\int_{\phi=0}^{\pi} L_{ss}(\phi, \infty, l)(\vec{u_c}.\vec{n}) sin(\phi) d\phi d\theta \quad (9)$$

Thanks to the symmetry on θ, we can choose the spherical coordinates centered on P such that

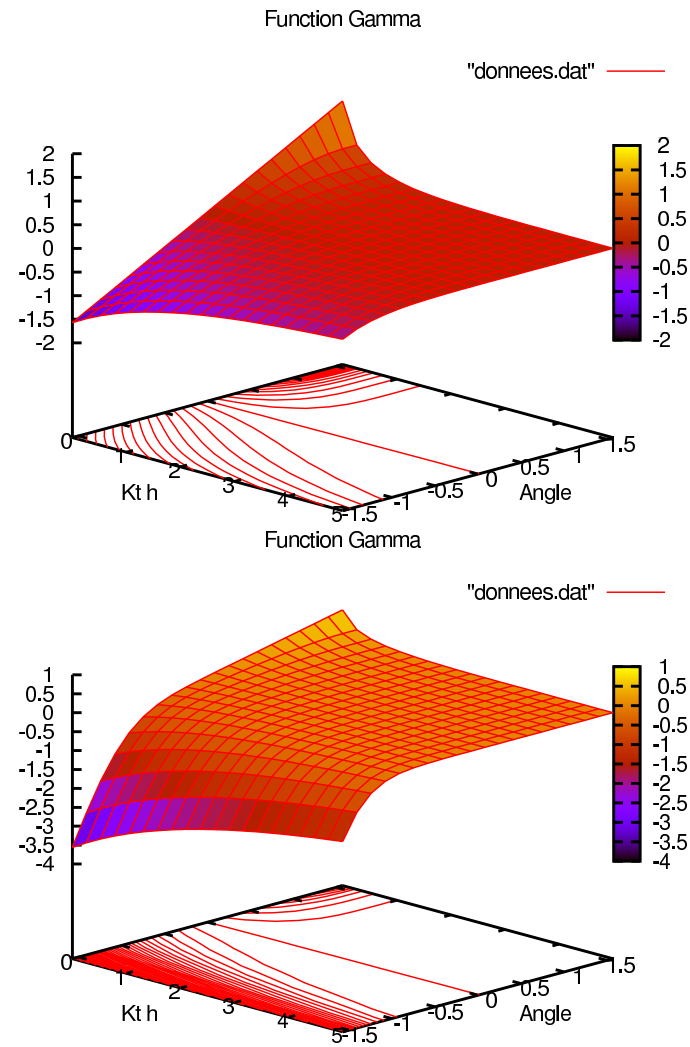

Fig. 3. The Γ function for isotropic and hazzy phase function

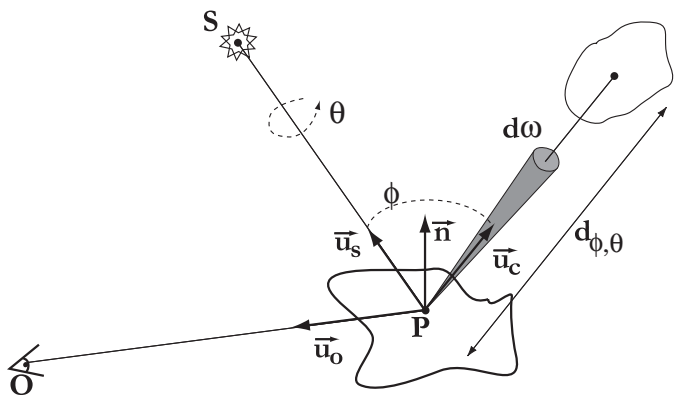

Fig. 4. Notations for Lambert illumination model

$\vec{n} = (sin(\phi_n), 0, cos(\phi_n))$ (cf.Figure 4) and write :

$$\begin{aligned} L_{iss}(P) &= k_d \int_{\phi=0}^{\pi} L_{ss}(\phi, \infty, l) \left[\int_0^{2\pi} cos(\phi_n) cos(\phi) d\theta \right. \\ &+ \left. \int_0^{2\pi} sin(\phi_n) sin(\phi) cos(\theta) d\theta \right] sin(\phi) d\phi \end{aligned}$$

It remains after integration :

$$L_{iss}(P) = 2\pi k_d \int_{\phi=0}^{\pi} L_{ss}(\phi, \infty, l) cos(\phi_n) cos(\phi) sin(\phi) d\phi$$

Now we can substitute L_{ss} with equation (4) to obtain :

$$L_{iss}(P) = 2\pi k_d cos(\phi_n) \int_{\phi=0}^{\pi} \frac{k_t^2 I_S \Omega}{4\pi l} g(l, \phi) cos(\phi) d\phi$$

with

$$g(\lambda, \phi) = \frac{1}{\lambda} \int_{\phi-\frac{\pi}{2}}^{\frac{\pi}{2}} p(\alpha + \frac{\pi}{2}) e^{-\lambda \frac{\cos(\alpha-\phi)+1}{\cos(\alpha)}} d\alpha$$

And finally we have :

$$L_{iss}(P) = \frac{k_t^2 \Omega I_S}{4\pi} \frac{2\pi k_d cos(\phi_n)}{k_t l} \Gamma_L(k_t l) \quad (10)$$

with

$$\Gamma_L(\lambda) = \int_0^{\pi} cos(\phi) \int_{\phi-\frac{\pi}{2}}^{\frac{\pi}{2}} p(\alpha + \frac{\pi}{2}) e^{-\lambda \frac{\cos(\alpha-\phi)+1}{\cos(\alpha)}} d\alpha d\phi \quad (11)$$

The Phong model is much more difficult and we need the reparametrization of Ramamoorthi et al. [24]. It gives use the following equation :

$$L_{iss}(P) = k_s \int_{\theta=0}^{2\pi}\int_{\phi=0}^{\pi} L_{ss}(\phi', \infty, l) cos^n(\phi) sin(\phi) d\phi d\theta$$

where n is the shininess and k_s the specular reflectivity of the surface. ϕ and θ are defined relatively to the axe $\vec{u_r}$. ϕ' is the angle formed by vector $\vec{u_s}$ and $\vec{u_c}$ (see Figure 5). We can choose the base to have :

$$\phi' = acos(cos(\phi_S) cos(\phi) + sin(\phi) cos(\theta) \sqrt{1 - cos^2(\phi_S)})$$

Therefore, the equation will be :

$$L_{iss}(P) = \frac{k_t^2 \Omega I_S}{4\pi} \quad \frac{k_s}{k_t l} \quad \Gamma_P(k_t l) \tag{12}$$

with

$$\Gamma_P(\phi', \lambda) = \int_0^{2\pi} \int_0^{\pi} \int_{\phi' - \frac{\pi}{2}}^{\frac{\pi}{2}} p(\alpha + \frac{\pi}{2}) e^{-\lambda \frac{\cos(\alpha - \phi') + 1}{\cos(\alpha)}} cos^n(\phi) d\alpha \tag{13}$$

D. Shadows on view rays

We want to integrate now the effect of occlusions along any view rays. Equation (7) describes the particular case where the view ray remains totally lit. In order to integrate shadow volumes, we need to consider more general cases, illustrated in Figure 6. Indeed, the ray must be split into lit and shadowed parts. In this example, the medium contribution along the ray is split into two parts on OA and BP. Using the laplace formula, it is straightforward to see that equation (7) becomes :

$$\begin{aligned} L_{dss}(d, \phi) &= \frac{k_t^2 \Omega I_S}{4\pi} e^{-k_t t} [\Gamma(k_t h, \gamma_P) - \Gamma(k_t h, \gamma_B) \\ &\quad + \Gamma(k_t h, \gamma_A) - \Gamma(k_t h, \gamma_O)] \end{aligned} \tag{14}$$

IV. Hardware implementation

A. Overview of our method

Our algorithm is easy to implement. We present here the 5 steps of this method and we will precise for each step if the computation is done by CPU or by GPU.

1) (CPU) The silhouettes of every moving shadow caster are computed. If the light source is moving, every silhouette needs to be recomputed.
2) (GPU) Scene is rendered using the conventional polygonal rendering method to obtain the direct illumination and the indirect single scattering. Surface shadows can be obtained using shadow planes algorithms [20], [25]. The stencil buffer now contains lit areas of the scene. An ambient fog is added to take into account both absorption and multiple scattering.
3) (GPU) Scene is rendered once more time and direct single scattering is computed for each vertex of the scene. Depth test is set to equality. Only lit parts of the scene are rendered thanks to the stencil buffer.

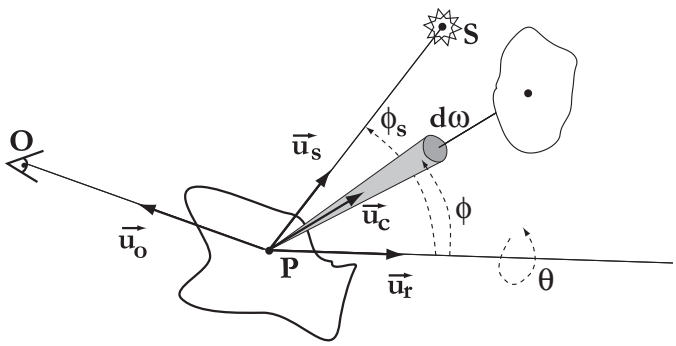

Fig. 5. Notations for Phong illumination model

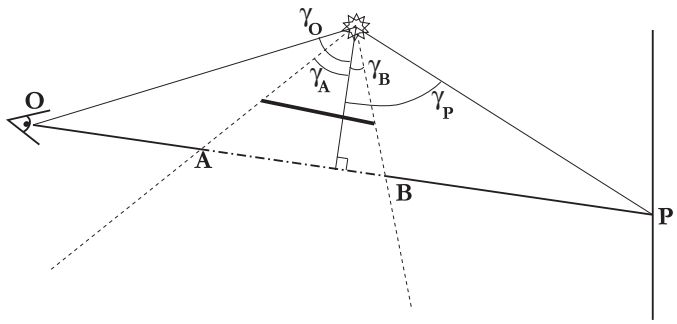

Fig. 6. Case of a partially shadowed view ray

4) (CPU) Shadow planes determined by the object's silhouettes are sorted in a back to front order.
5) (GPU) Shadow planes are rendered in that precise order. The depth test function accepts only planes that are closer to the camera. Front facing planes add their contribution when back facing planes subtract them. Stencil function is set to allow fragments if the stencil is equal to 1 for front facing planes and 0 for back facing ones. Front facing planes always decrement the stencil buffer and back facing ones always increment it.

All stages have to be done for each light source. As in [25], a initialization stage is done to obtain the ambiant lighting and the first depth map. Each stage is detailed in the following sections.

B. Computation of silhouettes (step 1)

In our algorithm, we select some objects to be shadow casters. Their silhouettes are easily computed by determining all edges of their mesh common to a front-facing triangle regarding the light position and one back facing it. Then all these edges are linked together if possible, and stored in a loop list. In order to obtain correct silhouettes, we need well designed closed triangular meshes (2-manifold) for which connectivity information are available. These conditions for the shadow casters are the ones indicated in [25]. Shadow planes are infinite quads formed by a silhouette edge and the light position. They are constituted by the two edge's vertices and two other points, projections of the previous vertices to infinity toward direction : light position - vertex [20]. They are oriented toward the unshadowed area of the scene. As we need to compute the medium contribution on all shadow planes, it is wise to use shadow plane silhouettes rather than the shadow planes of all little triangles. Of course, if the light does not move, only moving shadow caster silhouettes have to be computed. Finally, if the input geometry is modified by graphics hardware, using displacement mapping for example, a solution to obtain silhouettes of all objects quickly and accurately can be found in [26].

C. Rendering the scene (step 2 and 3)

The second and third steps of the algorithm compute the illumination on surfaces. This can be done using one or

```
varying vec4 posInScene;
// coeff_milieu.rgb = light_intensity.rgb*albedo.rgb*coef_extinct^2/4*PI
// coeff_milieu.a  : extinction coefficient
uniform vec4 coeff_milieu;
uniform vec4 poslght; // Light position
// integralRange (min,max,max-min)
uniform vec4 integralRange;
uniform sampler1D monarctan;
uniform sampler2D integrale;
uniform sampler2D depthtex;

void main() {
  vec4 facteurs,resultat;
  float rayDist ;
  vec3 rayDir;
  // facteurs.x = -t
  // facteurs.y = h
  // facteurs.z = alpha0
  // facteurs.w = alphad

  // View ray computation
  rayDir.xyz = posInScene.xyz;
  rayDist = length(rayDir);
  rayDir /= rayDist;

  // -t computation
  facteurs.x = -dot(poslght.xyz,rayDir);
  // h computation
  facteurs.y = length(poslght.xyz + rayDir * facteurs.x);

  // alpha0 et alphad computed thanks to atan texture
  facteurs.z = (facteurs.x/(facteurs.y*20.0)) + 0.5;
  facteurs.z = (texture1D(monarctan,facteurs.z)).r;  // atan(-t/h)
  facteurs.w = (rayDist+facteurs.x)/(facteurs.y*20.0) + 0.5;
  facteurs.w = (texture1D(monarctan,facteurs.w)).r;  // atan(d-t/h)

  // Ldss computation
  resultat.a = coeff_milieu.w*facteurs.y; // Kt h
  resultat.b = facteurs.z;                // alpha0
  resultat.r = texture2D(integrale,resultat.ab).r;
  resultat.b = facteurs.w;                // alphad
  resultat.g = texture2D(integrale,resultat.ab).r;
  resultat.rg = resultat.rg*integralRange.zz + integralRange.xx;

  // Final depth test
  rayDir.x = gl_FragCoord.x/1024.0;
  rayDir.y = gl_FragCoord.y/1024.0;
  rayDir.z = texture2D(depthtex,rayDir.xy).x;
  if (gl_FragCoord.z <= rayDir.z + 0.0001) {
    gl_FragColor.a = (resultat.g - resultat.r)/ facteurs.y;
    gl_FragColor.rgb = coeff_milieu.xyz * exp(-coeff_milieu.w * rayDist) * gl_FragColor.a;
  }
  else {
    discard;
  }
}
```

Fig. 7. Shader for L_{dss} computation

two rendering of the scene depending on the ability of the graphic cards to handle large shaders. Indeed, the number of instructions in the shader for both indirect single scattering and direct single scattering computations could exceed the capacity of the graphic card.

We show in Figure 7 and 8 the two GLSL fragment shaders used for steps 2 and 3 and which “implement” respectively equation (1) and (7). These shaders, and the corresponding steps, can be grouped together if possible. Note that we use a shader for any kind of participating medium of the scene. The two shaders need four textures : a 1D texture used as a lookup table for arctangent function, a 2D texture representing the function Γ defined in (6), 1D texture to store the Lambertian function (11) and finally one for the Phong function (13). Note that in equations (7), (10), (12) the first fraction could be a uniform variable and the second fraction could be computed directly by the shaders. Moreover we have avoided the direct computation of $atan$, cos or sin functions in these shaders,

```
varying vec3 lightDir,eyeVec,normal;
varying float att;
uniform vec4 coeff_milieu; // Same as the first shader
uniform sampler1D gamma_lambert;
uniform sampler2D gamma_phong;
// integralRange.x : Minimum value of Gamma L
// integralRange.y : Range of Gamma L
// integralRange.z : Minimum value of Gamma P
// integralRange.w : Range of Gamma P
uniform vec4 integralRange;

void main()
{
  // facteurs.r = I then kt * I
  vec4 facteurs,resultat;
  vec3 N = normalize(normal);
  facteurs.r = length(lightDir);
  vec3 L = lightDir/facteurs.r;

  float lambertTerm = dot(N,L);

  if(lambertTerm > 0.0) {
    // Standard computation of lambertian and phong illumination
    color += att * gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse * lambertTerm;

    vec3 E = normalize(eyeVec);
    vec3 R = reflect(-L, N);
    float dre = max(dot(R, E), 0.0);
    float specular = pow(dre,gl_FrontMaterial.shininess);
    color += gl_LightSource[0].specular * gl_FrontMaterial.specular * specular *att;

    // ISS lambert
    float dist = length(eyeVec);
    facteurs.r *= coeff_milieu.w;
    facteurs.g = texture1D(gamma_lambert,facteurs.r/10.0).r*integralRange.y+integralRange.x;
    resultat.a = 6.2831853*lambertTerm*facteurs.g/facteurs.r;
    resultat.rgb = gl_FrontMaterial.diffuse.rgb*resultat.a;

    // ISS phong
    if (dre>0.0) {
      facteurs.g = facteurs.r/10.0;
      facteurs.b = dre;
      resultat.a = texture2D(gamma_phong,facteurs.gb).r*integralRange.w+integralRange.z;
      resultat.rgb += gl_FrontMaterial.specular.rgb*resultat.a/facteurs.r;
      gl_FragColor.rgb = facteurs.gbb;//*resultat.a;
    }
    color.rgb += coeff_milieu.rgb*resultat.rgb*exp(-coeff_milieu.w * dist)*att;
  }
  gl_FragColor = color;
}
```

Fig. 8. Shader for L_{iss} computation

and preferred simple scalar products and texture lookups.

D. Sorting the shadow planes (step 4)

Before rendering all shadow planes, we have to make sure that we will not render shadow planes, or part of them, that are themselves in shadow. If we do not care about this problem, it will create artifacts we call shadow in shadows, shadow planes that are in shadow must not be rendered. Then, we render the shadow planes, back- or front-facing, in a ”back to front” order and use the stencil buffer to avoid the drawing of shadowed shadow planes.

E. Rendering the shadow planes (step 5)

It remains to render the shadow volumes, i.e. to take into account the equation (14). We still have the stencil we have obtained in the stage 2. Shadow planes are rendered in the order defined in the previous stage. The same pixel shader of Figure 7 is used for the shadow planes. The only difference is blending : front facing planes add their contribution when back facing planes subtract them.

Taking into account correctly the shadow in shadow problem requires the use of the stencil buffer. The key idea is to draw only the planes that constitute the boundary between

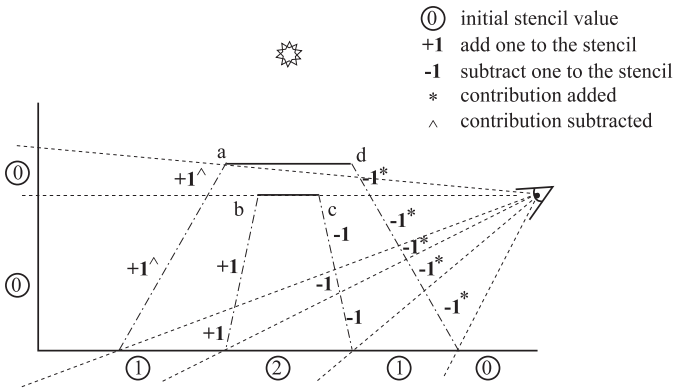

Fig. 9. Use of the stencil buffer in the rendering of shadow planes

lit volumes and shadowed ones. This can be done using our ordering and the value stored in the stencil. We define the stencil test such that front facing planes pass the stencil test if its value is one, representing shadowed area, and back facing ones passes if it equals zero, value representing lit area. Ideally the back (resp. front) facing quads should always add (resp. subtract) one to the stencil buffer if it passes depth test. Unfortunately, the stencil test is before the depth test so we have to do our own depth test in the pixel shader. Thus, we only draw shadow planes that make the shadow volume boundaries. The indicated strategy works if the camera is in the light. A slightly different strategy can be used when the camera is in shadow but the philosophy remains the same.

V. Results

The previous algorithm has been implemented on a standard computer using a 2.6 GHz processor and an ATI 9800 PRO graphics card (which is an old card now!). All images and videos we present have a 800x600 resolution. First of all, we point out, in Figure 10 the influence of each part of our single scattering illumination model. In this example, we have not considered the multiple scattering to concentrate on the other effects. On the top row, the first column shows the directe illumination of surfaces. The second column adds the direct single scattering. This effect highlights slightly all lit surfaces. Therefore, if we don't consider shadow volume, discontinuity in the illumination can occur between lit and shadowed areas, like in the bottom of the jug. After the shadow volumes rendering, illustrated in the third column, the discontinuities disapear. All these pictures have been rendered with indirect single scattering, that introduces subtle modifications : dimming of the specular highlights on the vase and brightening of darker region.

We also present some snapshots of our animations. The first image in Figure 11.a. is a simple scene, where a pen is bumping in front of two lights. It illustrates a classical situation where well design 3D objects are moving and casting shadows. Here the shadow planes are really intricated. This scene is rendered at more than 30 fps, including direct single scattering, indirect single scattering and of course volumetric shadows rendering. The image in Figure 11.b is a snapshot of an animation presenting two point light sources, including one that moves, in a complex scene containing about 100 000 triangles. The last picture 11.c is another snapshot from an animation where camera moves and properties of the participating medium evolve. The table II presents the FPS compared to the number of triangles of those scenes.

scene	without iss with dss and shadows	with iss, dss and shadows	number of triangles
fig. 11 .a	31	30	14 785
fig. 11 .b	10.5	10.2	110 014
fig. 11 .c	7.6	7	136 266

TABLE II

FPS COMPARED TO NUMBER OF TRIANGLES

VI. Conclusion

We have presented in this paper a complete single scattering illumination model along with a new algorithm able to render the main part of this model. It considers a single participating medium recovering the scene and lit by one or several, eventually dynamic, point light sources. Our algorithm is fast enough to handle more than 30 frames per second for moderately complex scenes. It implements the direct single scattering, the indirect single scattering and, most of all, volumetric shadows along with surfacic shadows. Our method can be implemented in programmable graphics hardware, and compared to volume rendering approach does not need a lot of texture memory. Therefore it can easily be integrated in any graphics engine. We also have to point out that our algorithm does not create the aliasing effect we can have with volume rendering techniques thanks to the use of the exact shadow planes.

The perspectives are numerous but two developments seems promising :

Relax assumptions : So far we consider only isotropic point light and homogeneous participating media. We have made satisfying preliminary work about animating several bounded participating media and it could be a way to handle heterogeneous medium. Directional light source are easier to handle than point light and will not be a problem. The real challenge is the integration of directionnal point light sources.

Make it soft ... and quick : An other way to obtain more realistic image will be to consider soft shadows, for surfacic shadows but also for volumetric shadows. Fortunately, our work is well adapted to algorithm such as [27] since it uses shadow volumes. Finaly, note that no optimization have been done to render the scenes. We believe that occlusion culling, frustum culling and clustering would greatly speed up our algorithm.

Acknowledgment

We thank Pascal Lecocq for the developments done in his thesis about this subject and wish him a excellent career in Spain. We also thank Leonie Van Royeen for her english review.

Fig. 10. Insight of the influence of the different contributions. The first column is the result after step 2, the second presents the addition of the direct single scattering and the third adds the volumetric shadows

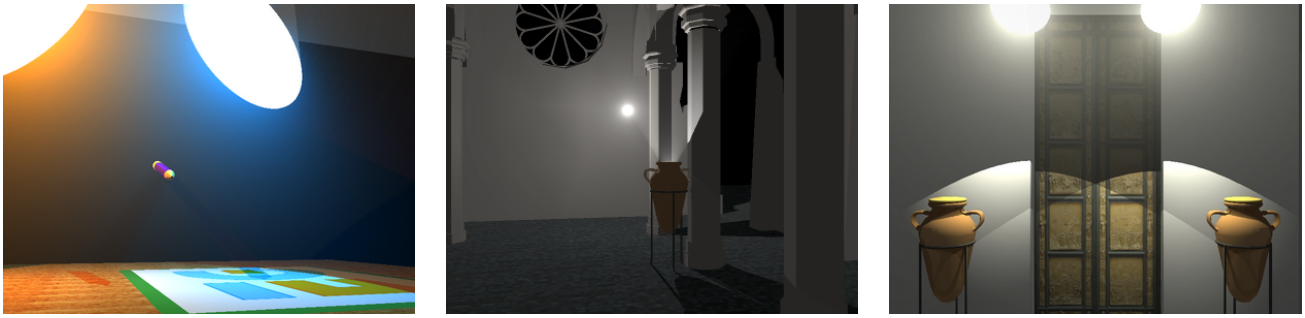

Fig. 11. Some snapshots of our animations. a. (left) a simple scene with a pen flying between two lights. b. (center) A complexe scene with two lights c. (right) A more complex scene

REFERENCES

[1] J. F. Blinn, "Light reflection functions for simulation of clouds and dusty surfaces," in *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1982, pp. 21–29.

[2] B. Sun, R. Ramamoorthi, S. Narasimhan, and S. Nayar, "A practical analytic single scattering model for real time rendering," in *proceedings of SIGGRAPH'05, Computer Graphics*, vol. 24 (3), 2005, pp. 1040–1049.

[3] U. Behrens and R. Ratering, "Adding Shadows to a Texture-based Volume Renderer," in *proceeding of 1998 Symposium on Volume Visualization*, 1998, pp. 39–46.

[4] R. Westermann and T. Ertl, "Efficiently Using Graphics Hardware in Volume Rendering Applications," in *proceeding of SIGGRAPH'98, Computer Graphics*, 1998, pp. 169–177.

[5] J. Stam, "Stable Fluids," in *proceedings of SIGGRAPH'99, Computer Graphics*, 1999, pp. 121–128.

[6] R. Fedkiw, J. Stam, and H. Jensen, "Visual Simulation of Smoke," in *proceedings of SIGGRAPH'01, Computer Graphics*, Aug. 2001, pp. 15–22.

[7] M. Nulkar and K. Mueller, "Splatting with Shadows," in *proceedings of Volume Graphics 2001*, 2001, pp. 35–49.

[8] R. Mech, "Hardware-Accelerated Real-Time Rendering of Gaseous Phenomena," in *Journal of Graphics Tools*, vol. 6(3), 2001, pp. 1–16.

[9] N. Hoffman and A. Preetham, "Rendering Outdoor Light Scattering in Real Time," 2002.

[10] S. O'neil, "Accurate atmospheric scattering," in *GPU Gems 2*, A. Wesley, Ed., Mar. 2005.

[11] Y. Dobashi, K. Kandea, H. Yamashita, T. Okita, and T. Nishita, "A Simple, Efficient Method for Realistic Animation of Clouds," in *proceedings of SIGGRAPH'00, Computer Graphics*, Aug. 2000, pp. 19–28.

[12] Y. Dobashi, T. Yamamot, and T. Nishita, "Interactive Rendering Method for Displaying Shafts of Light," in *proceeding of Pacific Graphics 2000*, 2000, pp. 31–37.

[13] C. Everitt, "A Fast Algorithm for Area Light Source Using Backprojection," 2002, www.r3.nu/cass/shadowsandstuff.

[14] S. Lefebvre and S. Guy, "Volumetric Lighting and Shadowing NV30 shader," 2002, lefebvre.sylvain.free.fr/cgshaders/vshd/vshd.html.

[15] Y. Dobashi, T. Yamamot, and T. Nishita, "Interactive Rendering of Atmospheric Scattering Effects Using Graphics Hardware," in *proceeding of Graphics Hardware 2002*, 2002.

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

[17] N. Max, "Efficient Light Propagation for Multiple Anisotropic Volume Scattering," in *proceedings of 5th Eurographics Workshop on Rendering*, 1994, pp. 87–104.

[18] N. Foster and D. Metaxas, "Modeling the motion of a Hot, Turbulent Gas," in *proceedings of SIGGRAPH'97, Computer Graphics*, Aug. 1997, pp. 181–188.

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

[20] T. Heidman, "Real Shadows Real Time," in *IRIS Universe*, vol. 18, 1991, pp. 28–31.

[21] V. Biri, D. Arquès, and S. Michelin, "Real Time Rendering of Atmospheric Scattering and Volumetric Shadows," in *Journal of WSCG'06*, vol. 14(1), Feb. 2006, pp. 65–72.

[22] 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," in *proceedings of Pacific Graphics 2000*, 2000, pp. 400–401.

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

[24] R. Ramamoorthi and P. Hanrahan, "Frequency space environment map rendering," in *proceedind of SIGGRAPH '02*, vol. 21(3). New York, NY, USA: ACM Press, 2002, pp. 517–526.

[25] C. Everitt and M. Kilgard, "Practical and Robust Shadow Volumes," 2003. [Online]. Available: http://developer.nvidia.com/object/robust_shadow_volumes.html

[26] S. Brabec and H. Seidel, "Shadow Volumes on Programmable Graphics Hardware," in *proceedings of Eurographics'03*, vol. 22(3), 2003.

[27] U. Assarson and T. A. Mller, "A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware," in *proceedings of SIGGRAPH'03, Computer Graphics*, vol. 22(3), 2003, pp. 511–520.