Real-time Volumetric Lighting in Participating Media

Balázs Tóth and Tamás Umenhoffer,

Budapest University of Technology and Economics, Hungary

Abstract

Simulating light scattering in participating media, such as dust, fog or smoke, can greatly improve the overall realism of the images. This volumetric effect has been well studied in the context of off-line rendering but is still challenging for interactive applications. In this paper we present a GPU-based algorithm to compute volumetric light-shafts generated by single scattering in participating media. The proposed method uses shadow maps to account for shadowing and interleaved sampling to maintain high frame rates without sacrificing image quality.

1. Introduction

Volumetric scattering can greatly improve the overall realism of rendered images. The main idea behind these simulations is that the space between the visible surface and the camera is filled with tiny particles that absorb and scatter light photons colliding with them. The exact computation of scattering is hard as taking into account multiple light bounces between particles yields to a strong dependence between incoming and outgoing radiance at a particle. However in many cases multiple scattering does not contribute too much to the overall appearance of the final image, and computing single scattering (one light bounce only) can capture the main characteristics of light transport. We propose a fast method to calculate single scattering in homogenous media taking into account solid dynamic occluder objects and dynamic light sources.

2. Previous work

Volumetric effects are well known in the context of off-line rendering. Typical renderers use *ray marching* [KH84] for illumination accumulation and some kind of Monte Carlo quadrature [Rus94] or volumetric photon maps [JZJ08] to account for scattering. These methods require expensive computations, which restricts their application in interactive systems.

As the performance of graphics processors (GPU) grows, researchers have proposed new methods for interactive volumetric lighting effects. These techniques commonly fall into two categories, they can be based either on shadow volumes or on ray marching. Methods using shadow volumes [BAM06, Jam03] identify the shadowed parts of the eye rays via silhouette polygons extruded from the light source. These shadow volumes are rendered in back to front order, which has an additional sorting cost. Ray marching methods accumulate scattering along eye rays by advancing on the ray in small steps and updating the accumulated parameters iteratively. In real-time rendering a slice based volume rendering technique can be used to provide efficient per pixel ray marching where shadowing of solid objects can be computed with shadow mapping [DYN02,Mit04]. There also exist methods that combine shadow volumes and ray marching to speed up the classical ray marching algorithm [WR08]. Finally, it is also possible to use an image-processing radial blur operator to visualize light-shafts. These techniques work only if the light source is visible from the camera [Mit07].

Our approach is based on ray marching and is implemented in a single fragment shader as a post process. Solid object shadows are handled via shadow maps.

3. Volumetric lighting in participating media

Let us consider a ray of equation $\vec{x}(s) = \vec{x}_0 + \vec{\omega}s$, defined by origin $\vec{x}_0$, direction $\vec{\omega}$, and ray parameter s. The change of radiance $L(\vec{x},\vec{\omega})$ along this ray in non-emissive homogeneous participating media is expressed by the *radiative transport equation* [SKSS08]:

$$\frac{dL(\vec{x}(s),\vec{\omega})}{ds} = -\tau L(\vec{x}(s),\vec{\omega}) + \tau a \int_{\Omega'} L(\vec{x}(s),\vec{\omega}')P(\vec{\omega}',\vec{\omega})\,d\omega'.$$

where τ is the *density* describing the probability of collision in a unit distance, a is the albedo that equals to the

probability of scattering (i.e. not absorbing) after collision, and $P(\vec{\omega}', \vec{\omega})$ is the *phase function* describing the probability density of the scattering direction.

This integro-differential equation is difficult to solve since the unknown radiance appears in derivative, normal, and integrated forms. Such equations can be solved by Monte Carlo methods [SK08], but they are far too slow for real-time applications. Thus, we completely ignore multiple scattering and approximate the in-scattering integral assuming single scattering only. Let us denote the in-scattering term by L_i in the following way:

$$\tau a \int_{\Omega'} L(\vec{x}(s), \vec{\omega}') P(\vec{\omega}', \vec{\omega}) d\omega' \approx L_i(\vec{x}(s), \vec{\omega}).$$

Due to the simplifying assumption, L_i no longer depends on unknown radiance L. The resulting differential equation

$$\frac{L(\vec{x}(s), \vec{\omega})}{ds} = -\tau L(\vec{x}(s), \vec{\omega}) + L_i(s, \vec{\omega})$$

can be solved analytically (the correctness of the solution can be proven by inserting it into the differential equation):

$$L(\vec{x}(s), \vec{\omega}) = e^{-\tau s} L(\vec{x}_0, \vec{\omega}) + \int_0^s L_i(\vec{x}(l), \vec{\omega}) e^{-\tau(s-l)} dl.$$

The integral on the right hand side of the equation can be approximated with a finite Riemann summation:

$$L(x(s), \vec{\omega}) \approx L(\vec{x}_0, \vec{\omega}) e^{-\tau s} + \sum_{n=0}^{N} L_i(\vec{x}(l_n), \vec{\omega}) e^{-\tau(s-l_n)} \Delta l, \tag{1}$$

where the step size is $\Delta l = s/N$, i.e. it is proportional to the length of the ray and is inversely proportional to the number of sample points.

We should note here that the consecutive samples of the summation are independent which will be exploited during interleaved sampling described in Section 4.1.

Using the same argument for the incident radiance showing up in the in-scattering term, we get the following formula for L_i. If the scene has a single point light of power Φ, then only one $\vec{\omega}'$ direction needs to be taken into account, thus we have:

$$L_i(\vec{x}, \vec{\omega}) = \tau a \frac{\Phi}{4\pi d^2} v(\vec{x}) e^{-\tau d} P(\vec{\omega}_l, \vec{\omega}).$$

where d is the distance between the considered point and the light source and $\vec{\omega}_l$ is the direction of the light source from the sample point. Function $v(\vec{x})$ indicates the visibility of the sample point from the light source. It returns zero for sample points that are in shadow and one for lit sample points (Figure 1).

3.1. Scattering calculation with ray-marching

We can approximate the volumetric rendering equation with a ray marching method that iteratively evaluates equation 1.

Figure 1: *Taking into account solid geometry during ray marching*

Figure 2: *Computation of the scattered component by ray-marcing*

The algorithm executes the following steps (Figure 2):

1. In every pixel it determines the visible surface point and its reflected radiance that will be the boundary condition for the volume radiance.
2. It iterates along the ray from the surface to the camera making small steps. In a particular sample point on the ray
   - in-scattering term $L_i(x(l_n), \vec{\omega})$ is computed,
   - absorbtion factor $e^{-\tau(s-l_n)}$ from the sample point to the eye is obtained, and
   - their product is added to the accumulated radiance.
3. The accumulated radiance is stored in the pixel spearheaded by the ray.

4. GPU Implementation

For the calculation of the illumination term $L_i(l_n, \vec{\omega})$ we can use shadow mapping. Depending on the type of the light source we can generate either a normal 2D shadow map for spot lights or a cube map for point lights.

For the accumulation of the radiance while ray marching we can use the following shader code that regularly calls shadow test function `shadowMC` to get visibility indicator `v`:

```
L = L0 * exp(-s * tau);
for(float l = s - dl; l >= 0; l -= dl) {
    x += viewDir * dl;
    float v = shadowMC(shadowMap, x);
    float d = length(x);
    Lin = exp(-d * tau) * v * Phi/4/M_PI/d/d;
    Li = Lin * tau * albedo * P(x, viewDir);
```

```
    L += Li * exp(-l * tau) * dl;
}
```

This function initializes the radiance of the ray to the radiance of the surface at the beginning (`L0`) multiplied by the total absorbtion along the ray. Then the ray is marched by making steps of size `dl`. Ray marching is executed in light's space, that is in a coordinate system where the light is in the origin since this makes calculation simpler. In this space the distance of point $\vec{x}$ from the light source is $|\vec{x}|$ and the light direction is also parallel with $\vec{x}$. This direction is checked by shadow test. Working in light's space requires the transformation of the viewing direction in this space. The transformed viewing direction is denoted by `viewDir`.

Taking the view ray the sample point can be obtained by a single addition (`x`). By calling the shadow test function, we can determine whether or not the sample point is visible from the light source, i.e. whether scattering may happen here. Then the source power `Phi` is attenuated by the absorbtion resulting in incident radiance `Lin`. The incident radiance is multiplied by the albedo and the phase function and the result is accumulated to the ray radiance. Complicated phase functions like the Henyey-Greenstein phase function can be stored in a lookup table, and fetched by a single texture read.

The ray marching algorithm can be implemented as a post processing method. The necessary inputs of the method are the shadow map and a depth map taken from the camera to identify visible surface points. As these maps are usually present in the texture memory, the algorithm requires no additional special rendering passes. If multiple lights should be simulated, the post process can be run separately for each light sources, and their contribution can be added together.

4.1. Interleaved sampling

In the method discussed so far, ray marching evaluates N samples for every pixel, which would slow down rendering if N is high. However, if the number of sample points N is reduced, then smooth light-shaft boundaries are replaced by abrupt changes. One way of attacking this problem without sacrificing performance is the application of *interleaved sampling* [KH01]. Interleaved sampling exploits the fact that the volume lighting and the visible surfaces are similar at neighboring pixels, thus the information gained at a particular pixel can be well used in its neighbors as well.

Let us consider the sum of N terms of equation 1. We divide the screen into $M \times M$ pixel blocks, and the N terms are distributed in them in a periodic way. That is, in a particular pixel we evaluate just N/M^2 terms. If we assume that the samples in the $M \times M$ pixel blocks are lying approximately on the same view ray — which is reasonable if the visible surface points are close to each other — we can add their contributions together to obtain a solution similar to taking all N samples on the ray. This requires additional post processing passes to add the pixel contributions in a block together. As the addition filter is separable we can use an addition once in a horizontal and once in a vertical direction instead of taking all the $M \times M$ samples in a single filter operation.

5. Results

The method has been implemented in DirectX9 and tested on an NVidia 8800GTX GPU. The scene consists of 66000 triangles and is illuminated by a point light. Without volumetric scattering the application renders 110 frames per second at 800×600 resolution. With volumetric scattering but without interleaved sampling the performance drops to 42 frames per second. However, if we turn interleaved sampling on, the performance goes back to 100 frames per second again.

6. Conclusions

This paper proposed a light-shaft rendering algorithm running on the GPU. The algorithm uses shadow mapping to check whether or not a point in the participating media may be directly illuminated by the light source. Executing ray marching in light's space, the program is particularly simple. We also proposed the application of interleaved sampling that significantly increases the rendering speed while maintaining high image quality.

References

[BAM06] BIRI V., ARQUES D., MICHELIN S.: Real time rendering of atmospheric scattering and volumetric shadows. *Journal of WSCG 14* (2006), 65–72.

[DC07] DOMONKOS B., CSÉBFALVI B.: Interactive distributed translucent volume rendering. In *Winter School of Computer Graphics* (2007), pp. 153–160.

[DYN02] DOBASHI Y., YAMAMOTO T., NISHITA T.: Interactive rendering of atmospheric scattering effects using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2002), Eurographics Association Aire-la-Ville, Switzerland, Switzerland, pp. 99–107.

[Jam03] JAMES R.: True Volumetric Shadows. *Graphics Programming Methods* (2003).

[JZJ08] JAROSZ W., ZWICKER M., JENSEN H. W.: The beam radiance estimate for volumetric photon mapping. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes* (New York, NY, USA, 2008), ACM, pp. 1–112.

[KH84] KAJIYA J. T., HERZEN B. P. V.: Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), ACM, pp. 165–174.

[KH01] KELLER E., HEIDRICH W.: Interleaved sampling. In *Rendering Techniques 2001 (Proc. 12th Eurographics Workshop on Rendering* (2001), Springer, pp. 269–276.

[Mit04] MITCHELL J. L.: Light shaft rendering. In *ShaderX3: Advanced Rendering Techniques in DirectX and OpenGL*, Engel W., (Ed.). Charles River Media, Cambridge, MA, 2004.

[Mit07] MITCHELL K.: Volumetric light scattering as a post-process. In *GPU Gems 3*, Nguyen H., (Ed.). Addison-Wesley, 2007.

[Rus94] RUSHMEIER H.: Rendering Participating Media: Problems and Solutions from Application Areas. In *Proceedings of the 5th Eurographics Workshop on Rendering* (1994), pp. 35–56.

[SK08] SZIRMAY-KALOS L.: *Monte-Carlo Methods in Global Illumination — Photo-realistic Rendering with Randomization*. VDM, Verlag Dr. Müller, Saarbrücken, 2008.

[SKSS08] SZIRMAY-KALOS L., SZÉCSI L., SBERT M.: *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.

[WR08] WYMAN C., RAMSEY S.: Interactive volumetric shadows in participating media with single-scattering. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing* (2008), pp. 87–92.

Figure 3: *Sample scene without volumetric scattering on the left side and with scattering on the right side.*