Eurographics Symposium on Parallel Graphics and Visualization (2012)
H. Childs, T. Kuhlen, and F. Marton (Editors)

Light Propagation Maps on Parallel Graphics Architectures

A. Gruson[1], A. Hakke Patil[2], R. Cozot[1], K. Bouatouch[1], S. Pattanaik[2]

[1]IRISA, Rennes, France
[2]UCF, Orlando, United States

Abstract

Light going through a participating medium like smoke can be scattered or absorbed by every point in the medium. To accurately render such a medium we must compute the radiance resulting at every point inside the medium because of these physical effects, which have been modeled by the radiative transfer equation. Computing the radiance at any point inside a participating medium amounts to numerically solving this radiative transport equation. Discrete Ordinate Method (DOM) is a widely used solution method. DOM is computationally intensive. Fattal [Fat09] proposed Light Propagation Maps (LPM) to expedite DOM computation. In this paper we propose a streaming based parallelization of LPM to run on SIMD graphics hardware. Our method is fast and scalable. We report more than $20\times$ *speed improvement by using our method as compared to Fattal's original method. Using our approach we are able to render* $64 \times 64 \times 64$ *dynamic volumes with multiple scattering of light at interactive speed on complex lighting, and are able to render volumes of any size independent of the GPU memory capability.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional—Graphics and Realism I.3.3 [Computer Graphics]: Picture/Image—Generation

1. Introduction

Physically based rendering problems are challenging for research because of the complexity of computation, in particular, the rendering of participating media where light interacts with all the medium. Over the last few years progress in GPU technology has made a lot of parallel computing power accessible to us at a very low cost. Developers in every field have become interested in porting their algorithms to GPU to accelerate the computation. The main hurdle to this porting is that, GPUs are based on a SIMD architecture, and hence to run the ported algorithms efficiently, the CPU algorithms must be carefully transformed to take the best advantage of this SIMD architecture.

Light transport inside the participating medium has been modeled by the radiative transfer equation [Cha50]. There are several categories of numerical techniques to solve this equation. A particular category of techniques that is of interest to us is Discrete Ordinate Methods (DOM), which relies on the discretization of the volume and the directions, and solves the problem iteratively through local interactions. This approach is GPU friendly because GPU is efficient on regular and finite data.

In this paper we propose a GPU based DOM approach to solving the radiative transfer equation and rendering. Our work is based on Fattal's [Fat09] light propagation maps (LPM) approach. LPM is an efficient approach to solving light transport equation. However, because of the various constraints posed by GPU (for e.g.: parallel execution, branching condition, etc.), the LPM approach as proposed by Fattal cannot be directly ported to GPU. Using a novel data organization we transform the LPM approach to make it amenable to GPU implementation. Moreover, with a novel streaming mechanism we make the resulting algorithm scalable and hence capable of processing volumes of any size on a GPU independent of its memory capability.

This article is organized as follows. In section 2, we review existing methods on participating media rendering. In section 3 we examine the RTE equation, establish the notation and describe Fattal's approach. In section 4, we present our work on this technique. In the last sections, we show our results and discuss them.

DOI: 10.2312/EGPGV/EGPGV12/081-088

2. Previous works

There are several categories of numerical techniques to solve the RTE equation. We can find a survey on participating media rendering techniques as well as a classification in [CPCP*05, PCPS97]. Two main approaches exist for rendering with multiple scattering: deterministic and stochastic methods. In this paper we focus only on deterministic methods and particularly on the Discrete Ordinate Methods (DOM) [SH01].

Discrete Ordinate Methods These methods rely on the 3D discretization of space and direction. They solve the RTE iteratively through local interactions. However, the DOM techniques suffer from artifacts named false scattering and ray effect. Several techniques have been proposed to reduce these shortcomings. For example, the technique proposed by Languenou et al. [LBC94] uses ray casting to solve the boundary condition and to compute single scattering. Then, they use local interactions between voxels to compute the multi-scattering component. Fattal proposes another approach [Fat09] based on a fine sampling of the light propagation directions and a coarse sampling of the radiance stored for each voxel. We will explain that method in more detail in Section 3.

GPU-based algorithms Zhou et al. [ZRL*08] developed a technique achieving real-time animated smoke rendering including multiple scattering. Their approach is based on the decomposition of the input smoke animation into a sequence of points with a radial basis function and a residual field. They use low ordered harmonic spherics to store the lighting information. They handle real-time manipulation of viewpoint, smoke attribute and lighting. But they can not achieve fast smoke simulation due to the high preprocessing time needed to build their representation.

Multiple scattering can also be approximated by diffusion equation [Ish78] which consist of a 2 coefficient spherical harmonic expansion of the radiance field. This method was introduced in computer graphics by Stam [Sta95]. Bernabei et. al [BHPB*12] implemented a parallel lattice-boltzmann [GRWS04] solution of the diffusion equation for rendering heterogeneous refractive media. Szirmay-Kalos et. al [SKLU*09] accelerated the iterative solution to the diffusion equation by making an initial guess based on a homogeneous medium assumption. This method has been further extended in [SKLU*11]. Wang et. al [WWH*10] also implemented a parallel solution to diffusion equation however they used a tetrahedral mesh instead of cubic grids for representing the volume, allowing them to render arbitrarily shaped objects.

Englhardt et al. [END10] presented a very promising method which is based on the instant radiosity technique. They use VPL to compute single scattering and multiple scattering as well. Moreover, they need to clamp the VPL contribution to remove some artifacts. So, they use a compensation bias step to correct the clamping and to get good looking results. Their technique is fast and can allow a surface to contribute to the radiance of the participating medium. However, it does not handle complex light sources such as environmental maps.

3. Light transport in Participating media

The radiative transport equation (RTE) Eq. 1 models all the light interactions with a participating media. Light can be absorbed and/or scattered at every point in the volume:

$$(\omega\cdot\bigtriangledown)L(p,\omega) = Ka(p)L_e(p,\omega) - (Ka(p)+Ks(p))L(p,\omega) + \frac{Ks(p)}{4\pi}\int_{S^2} L(p,\omega_i)\rho(\omega,\omega_i)d\omega_i \quad (1)$$

where $L(p,\omega)$ is the radiance ($W\cdot m^{-2}\cdot sr^{-1}$) leaving a point p in direction ω. $L_e(p,\omega)$ is the emitted radiance, and is zero for non-self-emitting media. $Ka(p)$ and $Ks(p)$ are the absorption and scattering coefficients that characterize the volume. The right most term corresponds to multiple scattering where all the incoming directions are scattered in direction ω. $\rho(\omega,\omega_i)$ is the phase function that describes the angular distribution of radiance scattered along direction ω around a point when illuminated from a specific direction ω_i.

3.1. Fattal's algorithm

In [Fat09], Fattal proposed Light Propagation Map as a solution to the RTE equation. This method falls into the category of Discrete Ordinates Methods (DOM).

As done in most DOM methods, Fattal's method samples the spatial domain D into voxels C_{xyz} such that $\bigcup C_{xyz} = D$ where (x,y,z) are the voxel index. It also samples the unit sphere S^2 around the center of the voxel into a set of directions Ω^d so that the union of the solid angles around the sampled directions $\bigcup|\Omega^d| = S^2$. The goal of this technique is to approximate the radiance $L(p,\omega)$ (Eq. 2) by an average scattered radiance I^d_{xyz} in each voxel C_{xyz}.

$$I^d_{xyz} \approx (V^d_{xyz})^{-1} \int_{C_{xyz}}\int_{\Omega^d}\frac{Ks_{xyz}}{4\pi}\int_{\omega'\in S^2} L(p,\omega')\rho(\omega,\omega')d\omega' d\omega dp \quad (2)$$

where I^d_{xyz} is the radiance in the voxel xyz along the direction Ω^d. $V^d_{xyz} = \Delta_V|\Omega^d|$ is the product of the volume of the voxel C_{xyz} and $|\Omega^d|$ the solid angle. In this representation, the emission, absorption and scattering coefficients are assumed to be constant in each cell.

The main problem with the DOM techniques is that they suffer from two main shortcomings, namely false scattering and ray effect. The common solution to reduce these two artifacts is to increase the space and direction discretization resolution. However, this approach requires huge amount of memory and hence limits the method practically.

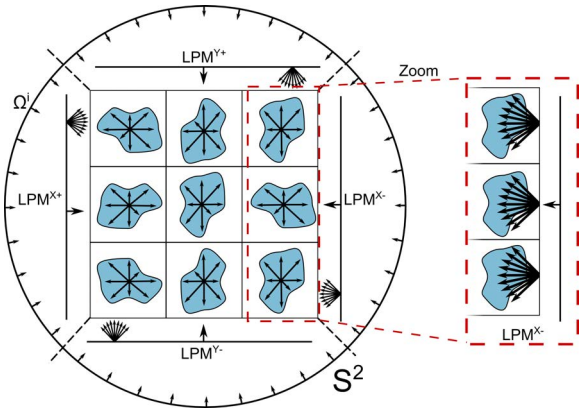

Figure 1: *The unit sphere S^2 is discretized into a set of m or n directions Ω^i. For each voxel, Fattal computes the scattered radiance for m directions with $m << n$, n being the number of propagation directions. An LPM is an array of point elements, each emitting light in n directions sampling solid angle $\frac{4\pi}{6}$ of the sphere S^2.*

To overcome this problem, Fattal proposes fine sampling of space and direction for the iterative light propagation and coarse sampling for storage. For the high resolution propagation, he uses 2D ray maps called Light Propagation maps, each of which is of high spatial resolution and represents a subset of the finely sampled propagation directions (Fig. 1). Starting with a face of the coarse grid (3D grid of voxels representing the medium), this 2D map propagates step by step in one voxel plane at a time and the propagation results are stored at the center of the voxels. So the LPM is basically a high resolution 2D array, each of its element (r,s) stores a set of rays originating at (r,s) and having directions sampled from the unit sphere S^2.

The propagation process must compute the interaction between the currently propagated LPM and the volume. For the first propagation, the energy carried by each ray is initialized with the boundary conditions. The boundary conditions correspond to the incoming radiance arriving at the volume boundary. Then the rays are marched using a grid marching algorithm along their directions. When a ray leaves the medium at the boundary point A, then a new ray of the same direction with zero radiance is generated from an opposite point B as shown in Fig. 2. The aim is to have the same number of rays in each part of the volume.

Let $L^{i,t}_{r,s}$ be the radiance of a ray originating at a point (r,s) of the LPM in direction Ω^i at the p^{th} propagation step from all the faces back and forth, t being the intersection number of the ray with the faces of the current traversed voxel. Let the voxel (x,y,z) be the t^{th} voxel encountered by the ray. The

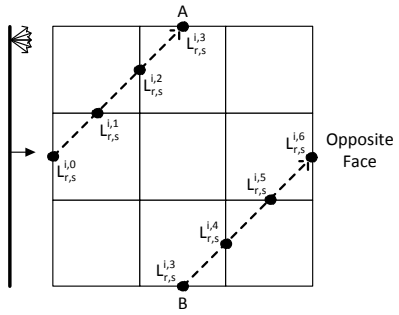

Figure 2: *$L^{i,t}_{r,s}$ is the ray radiance at the (r,s) element of the LPM, i the ray direction defined by Ω^i and t the number of the voxel traversed by the ray.*

radiance $L^{i,t}_{r,s}$ leaving the t^{th} voxel is given by

$$L^{i,t}_{r,s} = L^{i,t-1}_{r,s} e^{-\triangle l_q (Ka_{xyz}+Ks_{xyz})/\omega^n_z} + U^{d,p}_{xyz} \times (1 - e^{-\triangle l_q (Ka_{xyz}+Ks_{xyz})/\omega^n_z})/(Ka_{xyz}+Ks_{xyz}) \quad (3)$$

where $U^{d,p}_{xyz}$ is the stored unscattered radiance for direction d (equal to the closest propagation direction i), p is the p^{th} propagation (Scattering order). $\triangle l_q$ is the distance traversed by the ray in the voxel.

Given a propagation direction, when a ray adds its radiance contribution to the currently traversed voxel, this contribution must be scattered into all directions. Fattal stores the current scattering contribution in a variable named U that is used in future propagation steps. As we mentioned earlier, Fattal chooses different sampling frequency for I and U but for the reason of clarity, in the rest of the paper we will assume that they are the same. U and I are updated as:

$$\begin{aligned} R &= (V^d_{xyz})^{-1} A_{r,s} F^{i,d} L^{i,t}_{r,s} (1 - e^{-\triangle l_q Ks_{xyz})/\omega^n_z}) \\ I^d_{xyz} &= R \quad (Eq.2) \\ U^d_{xyz} &= R, \end{aligned} \quad (4)$$

where $A_{r,s}$ is the area of the (r,s) LPM sample. $F^{i,d}$ is precomputed as:

$$F^{i,d} = \frac{1}{4\pi} \int_{\Omega^i} \int_{\Omega^d} \rho(\omega, \omega') d\omega d\omega' \quad (5)$$

where superscripts i and d are for the propagation direction and storage direction respectively, "d" being the direction closest to the direction represented by "i".

Recall that one iteration represents the light propagation from each element (r,s) of the LPMs. It corresponds to one scattering while multiple iterations correspond to multiple scattering. Fattal proposes to terminate the iteration process when the unscattered light is low in every voxel. I and U are initialized with self-emission radiance (if any) of each voxel. Moreover in pratice, we keep only two U buffers, the

$(p-1)^{th}$ and the p^{th} U buffers and use a swap mechanism to reuse them.

Fattal's method is summarized by Algorithm 1.

Algorithm 1 Fattal's original algorithm

```
// p is iteration number (scattering order)
Initialize p_prev to 0, the index of the previous iteration
Initialize U_xyz^{d,p_prev} and I_xyz^d with medium emitted light
while max_xyz|U_xyz^{d,p_prev}| < ε && p>0 do
  // Initialisation U buffer of the p_cur^th iteration
  p_cur = (p_prev + 1)%2 // Current iteration index
  Initialize U_xyz^{d,p_cur} to 0
  for each LPM do
    for each propagation direction i do
      for each element (r,s) of the current map do
        if p == 0 then
          Initialize L_{r,s}^{i,0} with the boundary conditions
        else
          Set L_{r,s}^{i,0} to 0
        end if
        for each intersected voxel t do
          Update the ray radiance L_{r,s}^{i,t} from U_xyz^{d,p_prev}
          (Eq. 3)
          Update U_xyz^{d,p_cur} and I_xyz^d (Eq. 4)
        end for
      end for
    end for
  end for
  p_prev = p_cur
end while
```

4. New Method: Parallel and Scalable LPM

4.1. Parallelization

A simple strategy to parallelize Fattal's algorithm would be to assign a computation thread to each ray of the LPM. However, multiple rays affect the values stored in a voxel (see Fig. 3). So, this creates a synchronization problem. It may be possible to create synchronization barriers on the writable information and continue with the original simple approach. However, the algorithm will be less efficient because the latency, needed for the synchronization, will be significant as compared to the computation time.

In CUDA, updating the I and U voxel values could be performed using atomic operations on data of floating point type. However, using atomic operations has a significant cost because of the branching condition and the multiple global memory transactions (rather than cache accesses as in our approach thanks to data locality), as shown in Fig. 8.

We address this synchronization related problem by dividing the original propagation step into two steps: the propagation step and then the collecting step. These steps are illustrated in Fig. 4. In the propagation step, we group together all the rays of same propagation direction. Thus we guarantee that each voxel is traversed by only one ray at a time. So we create a temporary buffer to store the radiance brought by a ray when it goes through a voxel. In the collecting step, we use all the temporary buffers to update the $U_{xyz}^{d,p}$ and I_{xyz}^{d} values. In order to simplify the discussion, we assume that the LPM spatial resolution is the same as the volume face spatial resolution.

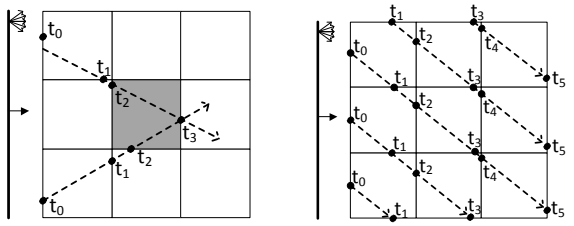

Figure 3: *On the left, we have the original Fattal's algorithm where we put one thread per ray. The problem is that it produces synchronization problem for writing into the grey voxel. On the right, we only propagate rays having the same direction, thus guarantee only one ray write per voxel.*

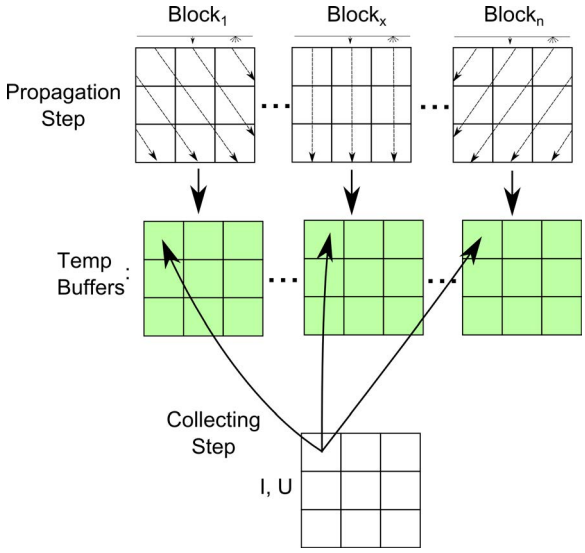

Figure 4: *Summary of the GPU algorithm. Each Block in the propagation step manages one temporary buffer. Then, in the collecting step, each voxel read temporary buffers to update I and U.*

We create two GPU kernels: one for the propagation step and the other for the collecting step. One propagation step corresponds to one LPM propagation and requires propagation of multiple blocks. Each block corresponds to all the elements of the LPM and to only one propagation direction.

There are as many blocks as directions. We assign a thread to each element (r,s) of the LPM. In this way, there is no execution divergence between threads in the same block. The role of a thread is to trace a ray with origin (r,s) and direction i and to compute its contribution to the traversed voxels. This contribution is stored into a temporary buffer. The ray contribution can be expressed as:

$$T^i_{xyz} += L^{i,t}_{r,s}(1 - e^{-\triangle l_q K s_{xyz}/\omega^n_z}), \quad (6)$$

where T^i_{xyz} is the temporary buffer value for the direction i in the voxel (x,y,z).

As for the collecting step, we create only one block representing all the volume (the whole participating medium). Each thread of this block is assigned to one voxel. It collects the result of the propagation step for all the directions.

Algorithm 2 Our algorithm

```
Initialize p_prev to 0, the index of the previous iteration
Initialize U^{d,p_prev}_xyz and I^d_xyz with self emitted radiance if any
// p = iteration number
for p ∈ [0, N_iterations[ do
  // Initialisation U buffer of the p_cur^th iteration
  p_cur = (p_prev + 1)%2 // Current iteration index
  Initialize U^{d,p_cur}_xyz to 0
  for each LPM do
    parallel for each blocks i do
      parallel for each thread (r,s) do
        Create ray and initialize L^{i,0}_{r,s}
        for each intersected voxel t
          Compute L^{i,t}_{r,s} from U^{d,p_prev}_xyz (Eq. 3)
          Store ray contribution into T^i_xyz (Eq. 6)
        end for
      end for
    end for
  end for
  Synchronize_blocks()
  // Collecting step
  parallel for each voxels (xyz) do
    for each direction i
      Update U^{d,p_cur}_xyz and I^d_xyz with T^i_xyz (Eq. 4)
    end for
  end for
  p_prev = p_cur
end for
```

4.2. Streaming

Memory limitation of GPUs severely restricts the size of volume we can handle at a time. Particularly, to run Algorithm 2 on a 256^3 volume with 25 propagation directions per one LPM face requires 3.8 GB memory. Maximum memory available on most of the GPUs is much less than this size. A solution is to stream portions of the volume to the GPU.

To this end, we split the volume V into sub-volumes B_{ijk} defined as

$$\begin{aligned} B_{ijk} = \{C_{xyz} |\ & x \in [i \times N_{sub}, (i+1) \times N_{sub}], \\ & y \in [j \times N_{sub}, (j+1) \times N_{sub}], \\ & z \in [k \times N_{sub}, (k+1) \times N_{sub}], \} \end{aligned}$$

where C_{xyz} is a voxel at the spatial position (x,y,z) and N_{sub} the size of a sub-volume such that $\bigcup B_{ijk} = V$. Then we group together sub-volumes into slices. This grouping is based on the orientation of the LPM face. For example, for an LPM face perpendicular to the Z axis, we group together sub-volumes into a slice $S_l = \{B_{ijk} | i \in [0, I_{max}], j \in [0, J_{max}], k = l\}$. In this way, the propagation iteration consists in $\frac{Z_{max}}{N_{sub}}$ steps where Z_{max} is the maximum value of coordinate z. Instead of applying Algorithm 2 to the whole volume, we apply it to one slice at a time. That means propagation and collection for one slice must complete before the next slice is processed by the GPU. The radiance propagated from the LPM through a slice is stored at the outgoing face of the slice and this stored radiance is in turn propagated through the next slice, and so on.

The subdivision into sub-volumes make efficient transfer of slices from the CPU memory to the GPU memory as explained in the next section.

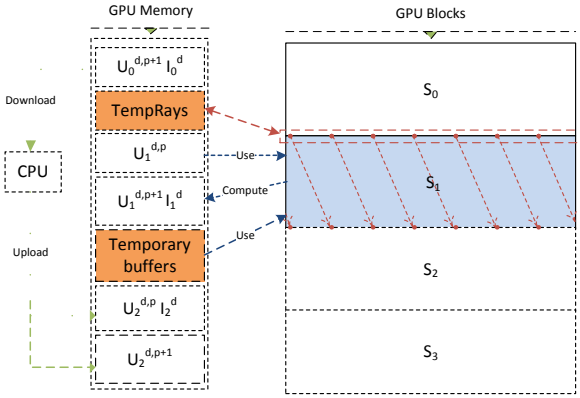

Figure 5: *S_i is the i^{th} slice with U_i and I_i the associated data. TempRays is the buffer which stores the radiance at the outgoing face of the previous slice S_0. This radiance are propagated within slice S_1 and finally deposited at the outgoing face of slice S_1. We use two streams represented by green and blue arrows. The green stream is in charge of upload/download data from the GPU memory to the CPU. The blue stream is in charge of the computation of the multiple scattering solution for the slice S_1. The orange buffers are the buffers that are kept in the GPU. Recall that d is the direction number and p the propagation iteration number.*

CUDA offers concurrent kernels execution and concurrent data transfers on modern GPU. We exploit this mecha-

nism by creating two streams: one to manage the GPU-CPU transfer, another to compute the multiple scattering solution for one entire slice (Fig 5). A stream is a FIFO queue of tasks. The role of the computation stream is to estimate the scattered light within slice S_i, while the transfer stream is in charge of downloading from the GPU the data (I,U) computed for the previous slice S_{i-1}. The computation and transfer streams run in parallel. As soon as the transfer stream completes downloading, its starts uploading the necessary data (I,U) for the next slice S_{i+1}.

In case our solution cannot fit the GPU memory, we split the slice into sub-slices and we stream each sub-slice to the GPU.

5. Implementation and Results

We used CUDA to implement our algorithm. We think that implementing it with OpenCL would be straightforward. We used two graphics cards: GTX 560M with 4 multiprocessors and 2 GB of memory, and GTX 580 with 16 multiprocessors and 1.5 GB of memory. We also implemented the original sequential version of Fattal's algorithm on a CPU i7 -2630QM running at 2GHz.

In our implementation, the user can specify the number of propagation directions for one LPM. Moreover, in case of isotropic phase function, the average radiance I is expressed for 1 direction while the unscattered radiance U is computed for 6 directions. However, in case of anisotropic phase function, the user can specify the number of directions for I and U.

The complexity of our algorithm is equal to $2spn^3$, where s is the scattering order, p the number of propagation directions and n^3 the number of voxels. Regarding the memory occupation, the complexity in bytes is equal to $4 * nbSpect * ((1+2U+I)n^3 + pn^2 + pn^3)$ without streaming and $4 * nbSpect * ((1+2U+I)Sn^2 + Spn^2 + pn^2)$ with streaming. U and I are the number of unscattered radiance directions and average radiance directions respectively. S is the slice width along the propagation direction and *nbSpect* the number of wavelengths (RGB in our case).

In order to guarantee only one thread per voxel, all the rays of same direction managed by a same block have to be synchronized whenever they leave their current voxel (Fig. 3). This is made possible by using a "__syncthreads()" instruction in their respective threads.

Note that the number of threads in one block is limited due to the number of registers available on a multiprocessor. This is why we reduce the size of a block by bringing down the resolution of one coordinate (r or s) of LPM, which increases the number of blocks for one propagation direction. Consequently, as several blocks may propagate light in a same direction, a voxel, lying at the frontier between two contiguous subsets R_i and $R_{(i+1)}$ of elements (r,s) of the current LPM, may be traversed by at least two rays, one coming

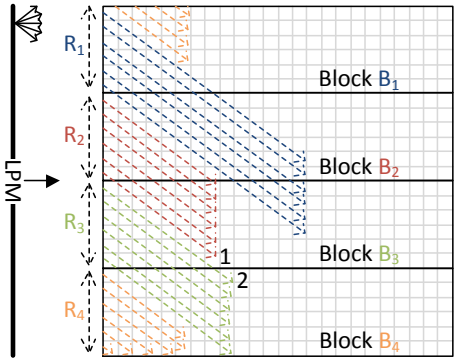

Figure 6: *Each block B_i is in charge of propagating rays from a subset R_i of elements (r,s) of a LPM. The rays of same directions at the border of the subset R_i and originating from two contiguous subset R_i may traverse at the same time a same voxel (for example rays 1 and 2).*

from R_i and the another from $R_{(i+1)}$ (Fig. 6). Each subset R_i is assigned one block B_i. Once again, to guarantee only one thread per voxel, we launch in parallel blocks B_{2i} then blocks $B_{(2i+1)}$. In this way, we avoid to launch contiguous blocks.

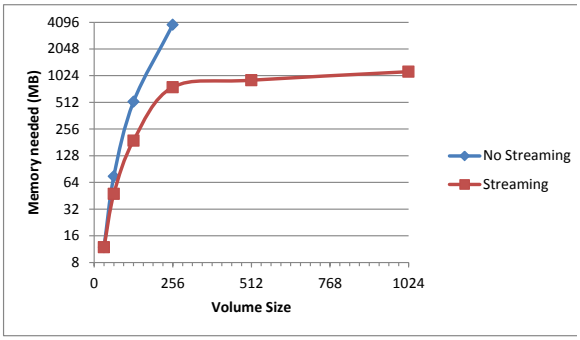

Figure 7: *GPU memory requirement for a 25 propagation directions, 6 storage directions (U and I). For the streamed technique, it is don't hit the memory bound of the GPU. Note that without streaming it is impossible to apply the algorithm on volume sizes larger than 256^3. For the streaming based technique we are far below the available amount of GPU memory. We must point to the fact that the memory requirement of streaming technique does increase with volume size (at a much slower rate though).*

In case the LPM resolution is N times higher than that of the voxel grid, to avoid the synchronization problem (due to the traversal of a same voxel by multiples rays) we divide the LPM into N smaller sub-LPM Sl_i and propagate each Sl_i one after the other.

Using $16 \times 16 \times 16$ voxels per sub-volume and 16×16 sub-volumes per slice, the memory needed by our algorithm is far below the available GPU memory (Fig. 7).

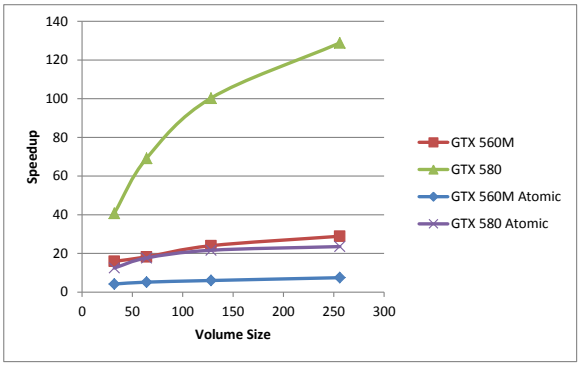

Figure 8: *Summary of the speedup between the original CPU algorithm and our implementation on 2 GPUs: GTX 560M and GTX 580. The green and red plots have been obtained with our parallel approach, while the blue and purple ones with atomic operations. We can observe that the obtained speedup increases with the volume size. This is due to the fact that a large number of blocks are created during the propagation step, which makes the GPU computing resources more and more busy. Note the efficiency of our approach compared to an atomic operation-based version.*

Furthermore, we use pinned memory as a buffer zone to speedup the data transfer between the CPU and the GPU. When the resulting data are downloaded from the GPU to the CPU, only one transaction is necessary to transfer all the data to the pinned memory. Next, we use *memcpy* operation to update sub-volumes on the CPU memory. Uploading the data onto the GPU memory is performed similarly.

For a volume of 64^3 voxels, 25 propagation directions and 3 bounce multiple scattering, our algorithm takes 417 milliseconds while the Fattal's algorithm running on the CPU takes 47 seconds. We see in Fig. 8 that the speedup grows with the volume resolution. This can be explained by the fact that the more the blocks, the better is the balancing of the multiprocessors loads. Examples of rendered images are given in Fig. 9, 10. Other results are shown in the accompanying video.

To validate our GPU-based parallel algorithm, we have computed the RMSE error between the CPU-based solution and our method. We found an RMSE of about 0.005.

6. Conclusions & Further works

We proposed a novel parallel algorithm to render participating media with multiple scattering. It is a parallel version of Fattal's algorithm. Compared to CPU based Fattal's algorithm, we obtained a speedup of 1 to 2 orders of magnitude.

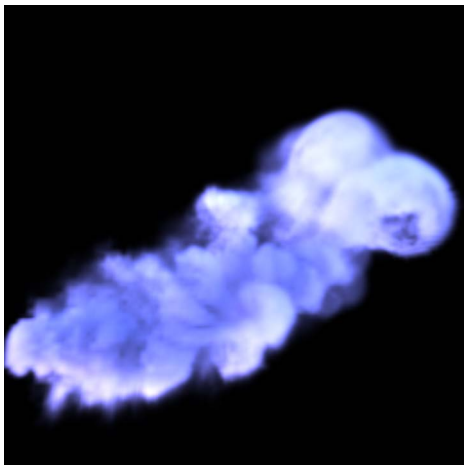

Figure 9: *Result of a* 128^3 *participating medium lit by a sky light source. It took 2291 ms to compute 3 scattering orders with 25 propagation directions.*

Figure 10: *Result of a* 128^3 *participating medium lit by an environmental map. It took 2291 ms to compute 3 scattering orders with 25 propagation directions.*

Our algorithm is capable of interactively rendering volumes of 64^3 voxels. We proposed a novel streaming technique based on the concept of sub-volume, slices, and sub-slices to handle any large size volume.

As a future work, our approach could use a hierarchical representation of participating media to handle huge volumes more efficiently. Moreover, a compression approach would allow to reduce the time needed for transferring data from the CPU to the GPU and vice versa. Finally, it would be interesting to extend our approach to handle solid objects inside participating media.

Acknowledgement

This work is partially supported by US National Science Foundation grant IIS-1064427.

References

[BHPB*12] BERNABEI D., HAKKE-PATIL A., BANTERLE F., BENEDETTO M. D., GANOVELLI F., PATTANAIK S., SCOPIGNO R.: A parallel architecture for interactively rendering scattering and refraction effects. *IEEE Computer Graphics and Applications 32* (2012), 34–43.

[Cha50] CHANDRASEKHAR S.: *Radiative Transfer*. Dover, 1950.

[CPCP*05] CEREZO E., PEREZ-CAZORLA F., PUEYO X., SERON F., SILLION F.: A survey on participating media rendering techniques. *the Visual Computer* (2005).

[END10] ENGELHARDT T., NOVAK J., DACHSBACHER C.: Instant multiple scattering for interactive rendering of heterogeneous participating media. *Technical paper* (2010).

[Fat09] FATTAL R.: Participating media illumination using light propagation maps. *ACM Trans. Graph. 28*, 1 (2009), 1–11.

[GRWS04] GEIST R., RASCHE K., WESTALL J., SCHALKOFF R. J.: Lattice-boltzmann lighting. In *Proceedings of the 15th Eurographics Workshop on Rendering Techniques, Norköping, Sweden, June 21-23, 2004* (2004), Keller A., Jensen H. W., (Eds.), Eurographics Association, pp. 355–362.

[Ish78] ISHIMARU A.: *Wave propagation and scattering in random media*. Academic Press, New York :, 1978.

[LBC94] LANGUENOU E., BOUATOUCH K., CHELLE M.: Global illumination in presence of participating media with general properties. *Proceedings du 5th Eurographics Workshop on Rendering* (1994).

[PCPS97] PEREZ-CAZORLA F., PUEYO X., SILLION F. X.: Global Illumination Techniques for the Simulation of Participating Media. In *Proceedings of the Eighth Eurographics Workshop on Rendering* (Saint Etienne, France, 1997).

[SH01] SIEGEL R., HOWELL J. R.: *Thermal Radiation Heat Transfer, 4th Revised edition*. Taylor & Francis Inc, 2001.

[SKLU*09] SZIRMAY-KALOS L., LIKTOR G., UMENHOFFER T., TÓTH B., KUMAR S., LUPTON G.: Parallel solution to the radiative transport. In *EGPGV* (2009), Debattista K., Weiskopf D., Comba J., (Eds.), Eurographics Association, pp. 95–102.

[SKLU*11] SZIRMAY-KALOS L., LIKTOR G., UMENHOFFER T., TOTH B., KUMAR S., LUPTON G.: Parallel iteration to the radiative transport in inhomogeneous media with bootstrapping. *IEEE Transactions on Visualization and Computer Graphics 17*, 2 (Feb. 2011), 146–158.

[Sta95] STAM J.: Multiple scattering as a diffusion process. In *In Eurographics Rendering Workshop* (1995), pp. 41–50.

[WWH*10] WANG Y., WANG J., HOLZSCHUCH N., SUBR K., YONG J.-H., GUO B.: Real-time rendering of heterogeneous translucent objects with arbitrary shapes. *Computer Graphics Forum (Proceedings of Eurographics 2010)* (2010).

[ZRL*08] ZHOU K., REN Z., LIN S., BAO H., GUO B., SHUM H.-Y.: Real-time smoke rendering using compensated ray marching. *ACM Trans. Graph. 27*, 3 (2008), 36.